

NEL SIS
THT - EL

HP 150 Personal Computer

HP 150 PROGRAMMER'S REFERENCE MANUAL

Included in
HP 150 Programmer's Tools
(Product 45435A)

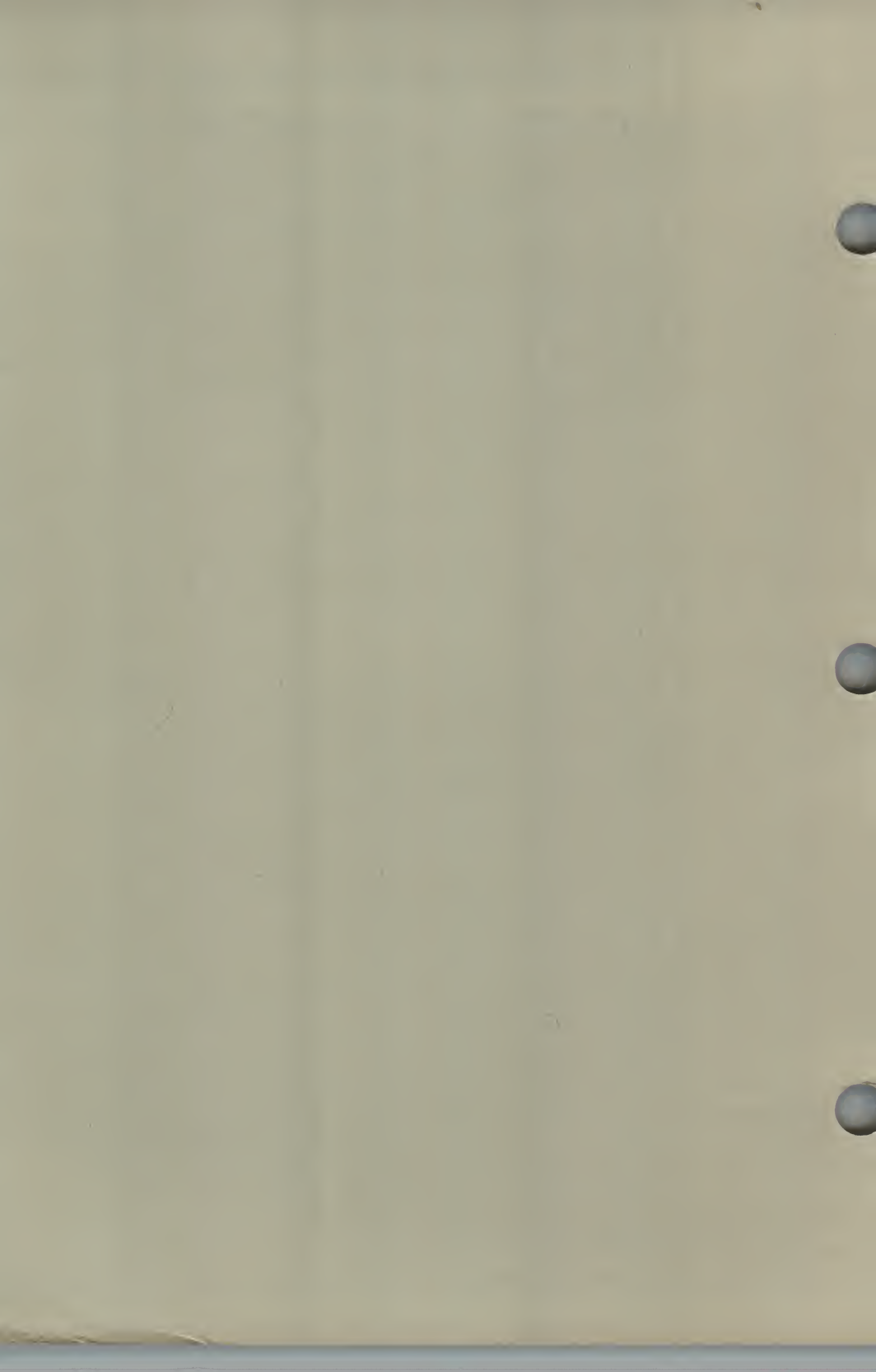


**HEWLETT
PACKARD**

974 EAST ARQUES AVENUE, SUNNYVALE, CA 94088

Part No. 45435-90002
(Not available separately)

Printed in U.S.A. 6/84



Reader Comment Sheet

HP 150 Personal Computer

October 1983

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

Is this manual technically accurate? Yes ☐ No ☐ (If no, explain under Comments, below.)

Are the concepts and wording easy to understand? Yes ☐ No ☐ (If no, explain under Comments, below.)

Is the format of this manual convenient in size, arrangement, and readability? Yes ☐ No ☐ (If no, explain or suggest improvements under Comments, below.)

Please indicate the title or part number of this manual: _____

Comments: _____

FROM:

NAME _____ TITLE _____

COMPANY _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS

PERMIT NO. 1355

SUNNYVALE, CALIFORNIA

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Personal Office Computer Division
P.O. Box 486
974 E. Arques Avenue
Sunnyvale, CA 94086

ATTN: Publications Manager

Reader Comment Sheet

HP 150 Personal Computer

October 1983

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

Is this manual technically accurate? Yes ☐ No ☐ (If no, explain under Comments, below.)

Are the concepts and wording easy to understand? Yes ☐ No ☐ (If no, explain under Comments, below.)

Is the format of this manual convenient in size, arrangement, and readability? Yes ☐ No ☐ (If no, explain or suggest improvements under Comments, below.)

Please indicate the title or part number of this manual: _____

Comments: _____

FROM:

NAME _____ TITLE _____

COMPANY _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1355 SUNNYVALE, CALIFORNIA

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
Personal Office Computer Division
P.O. Box 486
974 E. Arques Avenue
Sunnyvale, CA 94086

ATTN: Publications Manager

NELSYS

THT - EL

HP 150 Personal Computer

HP 150 PROGRAMMER'S REFERENCE MANUAL

Included in
HP 150 Programmer's Tools
(Product 45435A)



**HEWLETT
PACKARD**

974 EAST ARQUES AVENUE, SUNNYVALE, CA 94088

Part No. 45435-90002
(Not available separately)

Printed in U.S.A. 6/84

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

Copyright (c) 1984 by HEWLETT-PACKARD COMPANY
MS(TM)-DOS is a trademark of Microsoft Corporation
Microsoft (R) is a registered trademark of Microsoft Corporation
Intel (TM) is a trademark of Intel Corporation

TABLE OF CONTENTS

Section 1 - INTRODUCTION

About this Manual.....	1-1
Support.....	1-1
Compatibility.....	1-2
About the HP 150.....	1-2
Escape Sequence Programming.....	1-2
AGIOS Function Calls.....	1-3
Manual Overview.....	1-4

Section 2 - ESCAPE SEQUENCE PROGRAMMING

Introduction.....	2-1
Introduction to Softkeys.....	2-4
User-Definable Softkeys.....	2-4
Application Softkeys.....	2-4
Defining the User Softkeys.....	2-5
Displaying the User-Defined Softkeys.....	2-8
Displaying a User-Defined Message.....	2-9
Executing the User-Defined Softkeys.....	2-9
Programming Considerations.....	2-10
Using Touchscreen.....	2-11
Specifying Reporting Modes.....	2-11
Defining a Touch Field.....	2-12
Types of Touch Fields and Reporting.....	2-14
ASCII Fields.....	2-14
Keycode Fields.....	2-14
Toggle Fields.....	2-14
Normal Fields.....	2-15
Row/Column Reporting.....	2-15
Deleting Touch Fields.....	2-16
Changing Softkey Touch Sensitivity.....	2-17
Resetting the Touchscreen.....	2-17
Programming Considerations.....	2-18
Controlling the Keyboard.....	2-20
Transmit Function Key Mode.....	2-20
Programming Considerations.....	2-21
Controlling the Display.....	2-22
Display Enhancements.....	2-22
Selecting a Character Set.....	2-23
Alphanumeric Memory Control.....	2-25
Programming Considerations.....	2-26
Using the Keyboard and the Display.....	2-27
Two-Character Escape Sequence.....	2-27
Display Structure.....	2-31
Memory Addressing.....	2-31
Programming Considerations.....	2-33
Mode Selections.....	2-34
Mode Selection Escape Sequences.....	2-34
Programming Considerations.....	2-37
Format Mode.....	2-38

Defining Fields.....	2-38
Transferring Forms from the Screen to a Host Computer.....	2-39
Block Transfers.....	2-40
Handshaking.....	2-42
Correct Handshaking.....	2-43
Disable Auto Line Feed.....	2-43
The Appropriate Way to Receive Data.....	2-43
Data Operations.....	2-45
Selecting a Printer as the Destination Device.....	2-45
Selecting a Source Device.....	2-46
Paper Movement.....	2-47
Printer Modes.....	2-47
Record Mode.....	2-47
Data Logging Modes.....	2-47
Terminal to Printer Data Transfers.....	2-48
Copy a Selected Line.....	2-48
Copy Page.....	2-48
Copy All.....	2-48
Computer to Printer Data Transfers.....	2-48
Binary Data Transfer.....	2-49
ASCII Data Transfer.....	2-50
Data Transfer from HPIB Device.....	2-50
Determining Success of an Escape Sequence.....	2-51
Format Modes for the Internal Printer.....	2-52
Compressed Characters Mode.....	2-52
Report Format Mode.....	2-52
Metric Format Mode.....	2-52
Data Transfer Escape Sequences.....	2-53
Status Requests.....	2-58
Terminal Identification.....	2-58
Terminal Features.....	2-58
Primary Terminal Status.....	2-58
Secondary Terminal Status.....	2-59
Device Status.....	2-59
Cursor Position Sensing.....	2-59
Command Completion-Status.....	2-59
Status-Transfer Handshaking.....	2-60
Status-Transfer Priority.....	2-61
Terminal Identification.....	2-61
Interpreting the Status.....	2-62
Terminal Status.....	2-63
Primary Terminal Status.....	2-63
Secondary Terminal Status.....	2-67
Terminal Capabilities.....	2-69
Device Status.....	2-75

Section 3 - ALPHANUMERIC GRAPHICS INPUT/OUTPUT SUBSYSTEM

Introduction.....	3-1
How to Use AGIOS Function Calls.....	3-3
An Example on Calling an AGIOS Function.....	3-4
Syntax Used in AGIOS Function Calls.....	3-5

Section 4 - ALPHANUMERIC INPUT/OUTPUT SUBSYSTEM FUNCTION REFERENCE

Introduction.....	4-1
AIOS Functions.....	4-1

Batch Function Call.....	4-3
Video Intrinsic.....	4-5
Define Area.....	4-7
Write Area	4-9
Clear Area	4-12
Enhance Area.....	4-13
Read Area	4-14
Shift Area	4-16
Write Line	4-19
Application Softkeys.....	4-21
Update Softkey Label	4-22
Read Softkey Label	4-23
Display Softkey Labels	4-24
Control Functions.....	4-25
Execute Two-Character Sequence.....	4-26
Position Cursor.....	4-27
Define Enhancements.....	4-29
Cursor Sense Absolute.....	4-30
Cursor Sense Relative.....	4-31
Set Cursor Type.....	4-32
Read Fast AGIOS Entry Address.....	4-33
Go to Terminal Mode.....	4-34
Touchscreen Functions (ESC - z).....	4-35
Field Operations.....	4-35
ASCII Fields.....	4-35
Keycode Fields.....	4-35
Toggle Fields.....	4-35
Normal Fields.....	4-36
Row/Column Operations.....	4-37
General Procedure for Using Touchscreen.....	4-38
Define Touch Field.....	4-39
Define Softkey Field.....	4-41
Delete Touch Field.....	4-42
Touchscreen Reset.....	4-43
Set Touch Reporting Modes.....	4-44
Keyboard Processing.....	4-46
Key Characteristics.....	4-46
Normal.....	4-46
Intercept.....	4-46
Ignore.....	4-46
Beep.....	4-46
Special Keyboard Keys Definition.....	4-48
Keycode Table.....	4-52
Keycode Notes.....	4-54
Define Key Characteristics.....	4-55
Get Key Characteristics.....	4-57
Put Key.....	4-58
Keycode Mode.....	4-59
Using Keycode Mode.....	4-59
Defining Special Keys.....	4-60
Normal.....	4-60
Intercept.....	4-60
Ignore.....	4-60
Using Raw Mode.....	4-61
Turning On Keycode Mode.....	4-62
Reading Keycodes.....	4-63
Qualifier Word.....	4-63

Data Word.....	4-65
Touchscreen Input in Keycode Mode.....	4-66
ASCII Fields.....	4-66
Keycode Fields.....	4-66
Toggle and Normal Fields.....	4-66
Row/Column Reports.....	4-66
Softkeys in Keycode Mode.....	4-67
Turn Keycode Mode On/Off.....	4-68
Read Keycode Mode Status.....	4-69
Read Numeric/Graphics Keypad Status.....	4-70
Read Key.....	4-71

Section 5 - GRAPHICS INPUT/OUTPUT SYBSYSTEM FUNCTION REFERENCE

Introduction.....	5-1
GIOS Functions.....	5-1
Display Control (ESC * d).....	5-3
Clear Graphics Memory.....	5-5
Set Graphics Memory.....	5-6
Turn On Graphics Display.....	5-7
Turn Off Graphics Display.....	5-8
Turn On Alphanumeric Display.....	5-9
Turn Off Alphanumeric Display.....	5-10
Turn On Graphics Cursor.....	5-11
Turn Off Graphics Cursor.....	5-12
Turn On Rubber Band Line.....	5-13
Turn Off Rubber Band Line.....	5-14
Move Graphics Cursor Absolute.....	5-15
Move Graphics Cursor Incremental.....	5-16
Turn On Alphanumeric Cursor.....	5-17
Turn Off Alphanumeric Cursor.....	5-18
Turn On Graphics Text Mode.....	5-19
Turn Off Graphics Text Mode.....	5-20
Vector Drawing Mode (ESC * m).....	5-21
Select Drawing Mode.....	5-22
Clear Mode.....	5-22
Set Mode.....	5-22
Complement Mode.....	5-22
Jam Mode.....	5-23
Selective Erase.....	5-23
Select Line Type.....	5-25
Define Line Pattern and Scale.....	5-27
Define Area Fill Pattern.....	5-29
Fill Rectangular Area, Absolute.....	5-32
Fill Rectangular Area, Relocatable.....	5-33
Select Polygonal Fill Pattern.....	5-34
Select Boundary Pen.....	5-35
Set No Polygon Boundary.....	5-36
Set Relocatable Origin.....	5-37
Set Relocatable Origin to Current Pen Position.....	5-38
Set Relocatable Origin to Current Cursor Position.....	5-39
Graphics Text (ESC * m).....	5-40
Set Graphics Text Size.....	5-41
Set Graphics Text Orientation.....	5-44
Turn On Text Slant.....	5-45
Turn Off Text Slant.....	5-46
Set Graphics Text Origin.....	5-47

Display Graphics Text Label.....	5-48
Creating a Graphics Character.....	5-49
The Vector List.....	5-50
Building a Character.....	5-52
Re-defining the Entire Character Set.....	5-53
Programming Considerations.....	5-56
Define User Character Set.....	5-57
Select Default Character Set.....	5-58
Output Single Text Character.....	5-59
Set Graphics Defaults.....	5-60
Graphics Plotting (ESC * p).....	5-61
Current Pen Position.....	5-61
Absolute.....	5-61
Incremental.....	5-61
Relocatable.....	5-61
Lift Pen.....	5-63
Vector Move, Absolute.....	5-64
Vector Move, Incremental.....	5-65
Vector Move, Relocatable.....	5-66
Lower Pen.....	5-67
Vector Draw, Absolute.....	5-68
Vector Draw, Incremental.....	5-69
Vector Draw, Relocatable.....	5-70
Plot to Cursor Position.....	5-71
Point Plot.....	5-72
Set Relocatable Origin to Current Pen Position.....	5-73
Start Polygonal Area Fill.....	5-74
Terminate Polygonal Area Fill.....	5-76
Polygon Move, Absolute.....	5-77
Polygon Move, Incremental.....	5-79
Polygonal Move, Relocatable.....	5-80
Polygon Draw, Absolute.....	5-81
Polygon Draw, Incremental.....	5-82
Polygon Draw, Relocatable.....	5-83
Lift Boundary Pen.....	5-84
Lower Boundary Pen.....	5-88
Graphics Status (ESC * s).....	5-89
Read Pen Position.....	5-90
Read Cursor Position.....	5-91
Read Display Size.....	5-92
Read Graphics Settings.....	5-93
Read Graphics Text Status.....	5-95
Read Zoom Status.....	5-96
Read Relocatable Origin.....	5-97
Read Reset Status.....	5-98
Read Area Shading.....	5-99
Read Dynamic Graphics Capabilities.....	5-100
Set Picture Definition Defaults.....	5-101
Graphics Hard Reset.....	5-102
Read Extended Screen Dimensions.....	5-103

Section 6 - DATA COMMUNICATIONS PROGRAMMING

Introduction.....	6-1
Writing Your Own Data Comm Program.....	6-1
Assigning COM1 and COM2 Devices.....	6-2
Accessing Special Data Comm Functions.....	6-3

Data Comm Function Reference.....	6-4
Set 7-Bit Mode.....	6-6
Set 8-Bit Mode.....	6-7
Enable Data Comm Transparency Mode.....	6-8
Disable Data Comm Transparency Mode.....	6-9
Disconnect Modem.....	6-10
Send Break.....	6-11
Read/Write Data Comm Blocks.....	6-12
Send Data Block.....	6-13
Read Data Block.....	6-15
Programming Considerations.....	6-17
An Example of a Data Comm Program.....	6-19

Appendix A - AGIOS AND ESCAPE SEQUENCE QUICK REFERENCE

Appendix B - AGIOS FUNCTION CALLS FROM HIGH-LEVEL LANGUAGES

Getting Started with Pascal.....	B-2
Passing AGIOS Parameters from Pascal.....	B-3
Getting AGIOS Parameters from the Stack.....	B-5
Assembly Language Interface to Pascal.....	B-7
Passing Pointers to AGIOS from Pascal.....	B-10
Calling AGIOS from BASIC.....	B-12
A BASIC Program Calling AGIOS Functions.....	B-14

Appendix C - AGIOS FUNCTION CALL EXAMPLES

Define Area.....	C-2
Write Area.....	C-4
Clear Area.....	C-7
Enhance Area.....	C-9
Read Area.....	C-11
Shift Area.....	C-14
Write Line.....	C-17

Appendix D - GRAPHICS CONTROL FUNCTIONS

Introduction.....	D-1
Graphics Display.....	D-2
Keyboard Graphics Functions.....	D-3
Syntax Notations for Graphics Escape Sequences.....	D-5
Miscellaneous Characters.....	D-6
Control Codes.....	D-7
Graphics Display Control.....	D-7
Graphics Cursor Control.....	D-8
Graphics Memory Control.....	D-9
Graphics Drawing Mode Parameters.....	D-9
Drawing Modes.....	D-10
Drawing Patterns.....	D-12
Area Fills.....	D-16
Selecting an Area Fill Pattern.....	D-17
User-Defined Area Fill Patterns.....	D-17
Using Area Fill Patterns as Line Types.....	D-19
Rectangular Area Fills.....	D-20
Polygonal Area Fill.....	D-21
Area Boundary Pen.....	D-22

Graphics Relocatable Origin.....	D-22
Set RO, absolute.....	D-23
Set RO to Current Pen Position.....	D-23
Set RO to Graphics Control Position.....	D-23
Selecting Graphics Default Parameters.....	D-24
Graphics Hard Reset.....	D-24
Plotting Sequences.....	D-25
Plotting Commands.....	D-25
Lift Pen.....	D-27
Lower Pen.....	D-28
Use Cursor as Next Data Point.....	D-28
Rubber Band Line Mode.....	D-28
Draw a Point at Current Pen Position.....	D-29
Vectors.....	D-29
ASCII Format.....	D-29
Binary Format.....	D-32
Mixing Data Formats.....	D-35
Graphics Functions in Display Functions Mode.....	D-36
Graphics Hardcopy Operations.....	D-38
Initiating a Transfer from the Keyboard.....	D-39
Using the ESC&p Escape Sequences.....	D-39
Graphics Text.....	D-40
Keyboard Control of Graphics Text.....	D-41
Program Control of Graphics Text.....	D-43
Compatibility Mode.....	D-47
Compatibility Mode Configuration.....	D-49
Graphics Data.....	D-50
Graphics Data Format.....	D-50
Text.....	D-52
4014 Emulation.....	D-53
Programming Considerations.....	D-55
Graphics Status.....	D-58
Read Device ID (Parameter=1).....	D-59
Read Current Pen Position (Parameter=2).....	D-59
Read Graphics Cursor Position (Parameter=3).....	D-60
Read Cursor Position with Wait (Parameter=4).....	D-60
Read Display Size (Parameter=5).....	D-61
Read Device Capabilities (Parameter=6).....	D-61
Read Graphics Text Status (Parameter=7).....	D-63
Read Zoom Parameter (Parameter=8).....	D-63
Read Relocatable Origin (Parameter=9).....	D-64
Read Reset Status (Parameter=10).....	D-64
Read Area Shading Capability (Parameter=11).....	D-64
Read Graphic Modification Capabilities (Parameter=12).....	D-65
Any other Parameter.....	D-65
Graphics Control Escape Sequences.....	D-66
Graphics Display Control.....	D-66
Graphics Label Transmission.....	D-67
Vector Drawing.....	D-67
Plotting Commands.....	D-72
Graphics Status.....	D-73
Compatibility Mode.....	D-74

INTRODUCTION

SECTION

1

ABOUT THIS MANUAL

The *HP 150 Programmer's Reference Manual* is intended for the advanced programmer who is developing assembly-level application programs for the HP 150 system.

This manual assumes that you are familiar with high-level language programming and also with low-level Intel 8086/8088 assembly language programming. Because many of the procedures in this manual require assembly language programming and issuance of MS-DOS system function calls, the following documents are included with the manual:

- o MicrosoftTM Series 100/MS-DOS [®] User's Guide
- o Microsoft Series 100/MS-DOS Macro Assembler Manual
- o Microsoft Series 100/MS-DOS Programmer's Reference Manual
- o IntelTM iAPX 88 Book

The 3.5" disc that comes with this manual contains the following utilities:

Macro Assembler (MASM.EXE)
Linker (LINK.EXE)
Cross-Reference (CREF.EXE)
Library (LIB.EXE)
Line Editor (EDLIN.COM)
Debugger (DEBUG.COM)
Find (FIND.EXE)
File Compare (FC.EXE)
Sort (SORT.EXE)
File Conversion (EXE2BIN.EXE)

SUPPORT

Because of the specialized nature of this product and the many capabilities of the HP 150 when programmed at this level, assistance on using this product is provided only through consultation. Contact your local HP Sales and Service Office -- ask for Personal Computer Systems Engineering. Consulting is available by the hour (Product No. 45686A) or by the day (Product No. 45687A).

COMPATIBILITY

A hardware/software product designed to be dependent on one specific version of the HP 150 hardware, firmware, or software may be incompatible with other versions of the HP 150 -- or such a product may be incompatible with future personal computer products from Hewlett-Packard, including models based on the HP 150 itself.

ABOUT THE HP 150

The HP 150 supports two modes of operation: Terminal mode or Computer mode. It can function as a stand-alone personal computer and also as a full-function intelligent terminal.

Because the HP 150 is both a personal computer and an intelligent terminal, you can control the display and the keyboard in two ways:

1. via escape sequence programming, and
2. via Alphanumeric/Graphics Input/Output Subsystem (AGIOS) function calls.

This manual contains detailed information on developing applications using either escape sequences or AGIOS functions.

The HP 150 also provides an extended set of control functions for data communications, as well as the standard I/O functions of the MS-DOS operating system. This manual describes the extended data comm functions in detail, and explains how to use them to develop individual data comm programs.

Escape Sequence Programming

You can use escape sequences in standard console output statements in any programming language. When you send an escape character (ASCII code 27 decimal), the HP 150 terminal interprets the next one or more characters as special control codes, instead of as characters to be displayed on the screen.

However, the terminal's escape-sequence processor is relatively slow. Standard console output occurs at a rate of approximately 700 characters a second. If this speed is acceptable for your application, you will probably want to program with escape sequences.

AGIOS Function Calls

The second way to control the display and the keyboard is to use AGIOS function calls. The AGIOS, which is a unique feature of the HP 150, lets you write high-speed applications without using specific hardware information. Virtually all of the tasks that commonly require Basic I/O System (BIOS) calls can be done easily using AGIOS.

With AGIOS, your application need not depend on specific hardware addresses, revisions of computer firmware, or the Operating System BIOS.

In order to use AGIOS functions, your application must include special set-up conditions. For example, your applications must build an input buffer for the desired function code and any additional parameters. The desired function is executed via a software interrupt that is sent to the MS-DOS operating system.

The AGIOS function calls let you write high-speed output routines and gain total control over the HP 150 display and keyboard. AGIOS lets console output occur at rates up to 4000 characters per second.

AGIOS also gives you access to capabilities that are not accessible through escape sequences. These capabilities include intercepting keyboard keys, interpreting application softkeys, displaying a block of data on the screen, and creating user-defined graphics characters.

Thus, the use of escape sequence programming together with AGIOS function calls gives you a great deal of flexibility in developing application programs. If the desired operation is available via an escape sequence, and performance speed is not an issue, you may well prefer to use escape sequences. On the other hand, if your application requires faster console output, or if the function you want to use does not have an equivalent escape sequence, AGIOS is the preferred method. You can use escape sequences or AGIOS functions interchangeably depending on the nature of your application.

Introduction

MANUAL OVERVIEW

This manual consists of the following sections and appendices:

Section 1: Introduction

An overview of the material presented in this manual.

Section 2: Escape Sequence Programming

Programming with the escape sequences that are available in the HP 150.

Section 3: Alphanumeric/Graphics Input/Output Subsystem

Programming with Alphanumeric/Graphics Input/Output Subsystem (AGIOS) function calls. This section also includes the syntax used in the function reference in Sections 4 and 5.

Section 4: Alphanumeric Input/Output Subsystem Function Reference

Describes the Alphanumeric Input/Output Subsystem (AIOS) function calls. For each AIOS function, there is a detailed description of input parameters, output status, and additional remarks. For a few selected functions, an example is included to illustrate a possible use of the function.

Section 5: Graphics Input/Output Subsystem Function Reference

Describes the Graphics Input/Output Subsystem (GIOS) function calls. For each GIOS function, there is a detailed description of input parameters, output status, and additional remarks. For a few selected functions, an example is included to illustrate a possible use for the function.

Section 6: Data Communications

Writing data communications applications using standard MS-DOS Input/Output functions and Hewlett-Packard extended data communications functions. For each HP extended data communications function, there is a detailed description of input parameters, output status, and additional remarks. For some functions, an example is included to illustrate a possible use for the function.

Appendix A: AGIOS Functions and Escape Sequences Quick Reference

Lists all the AGIOS functions and the equivalent escape sequences, if available.

Appendix B: Calling AGIOS Functions from High-Level Languages

Information on calling AGIOS functions from high-level languages. The languages discussed in this appendix are Pascal and BASIC. The same principles apply to other languages.

Appendix C: Examples of AGIOS Functions

A Pascal-callable example of each video intrinsic in AIOS.

Appendix D: Graphics Control Functions

Information on the terminal's graphics functions and how they are used. The information and examples are intended for use in developing programs to control the graphics functions.

ESCAPE SEQUENCE PROGRAMMING

SECTION

2

NELSI THT - EL

INTRODUCTION

Escape Sequences provide a way to control the HP 150 keyboard and display via a program. A program sends an escape sequence to the terminal by printing the sequence to the console device. For example, the following BASIC statement tells the terminal to issue an 'ESC J' sequence to clear the display:

```
PRINT CHR$(27); "J"
```

When the HP 150 receives an escape sequence from an executing program, it performs the operation specified in the sequence just as if the sequence had been entered by the operator from the keyboard in Terminal mode. The operations performed range from controlling the appearance of the display screen (such as displaying text full-bright or half-bright) to redefining a function key that is to be used by an application program you are writing.

Some escape sequences consist of two characters; others consist of more than two characters. Two-character and multiple-character escape sequences perform different types of functions; and the rules for entering them are also different.

Two-character escape sequences generally correspond to functions performed by a single keystroke at the keyboard. For example, the horizontal TAB key at the left of your terminal keyboard corresponds to the two-character escape sequence

ESC I

which performs a TAB ("forward tab"). In two-character escape sequences, you must specify the escape code characters exactly. The following sequence

ESC i

using the lower-case parameter "i" performs a BACKTAB. In general, upper-case and lower-case parameters in two-character escape sequences have very different meanings to the system.

Multiple-character escape sequences generally refer to functions that require one or more parameters. Usually, the order of the parameters is not important. The first three characters of a multiple-character escape sequence define the type of function to be performed. Positioning the cursor to a particular column and row position on the screen requires a multiple-character escape sequence (ESC & a). For example, the sequence

Escape Sequence Programming

ESC & a 5 c 10 R

places the cursor at the sixth column of the eleventh row on the terminal screen. In this case, however, the sequence

ESC & a 10 r 5 C

performs exactly the same function. Although the order of the parameters in an escape sequence is not important, the case of the parameters is significant.

Because the number of characters in a given sequence depends upon the number of parameters, and because several parameters can be combined in one escape sequence, the terminal needs to know where a sequence ends. For this reason, the last parameter in a multiple-character escape sequence must be an upper-case letter, and all preceding parameters must be lower-case. The first upper-case letter following the ESC character ends the sequence.

You can combine multiple-character escape sequences of the same type in a single sequence. For example, if you want to change the values of the Local Echo and Caps Lock parameters in your Terminal Configuration menu, you can change individual values using the following sequences:

<u>Value:</u>	<u>Escape Sequence:</u>
Local Echo = NO	ESC & k 0 L
Local Echo = YES	ESC & k 1 L
Caps Lock = NO	ESC & k 0 C
Caps Lock = YES	ESC & k 1 C

To change both values, or to change any parameters in the ESC & k group, you could use one sequence. For example, to set Local Echo = No and Caps Lock = Yes, you could use either of the following sequences:

ESC & k 0 1 1 C
or
ESC & k 1 c 0 L

The final identifier must be an upper-case letter, and the preceding identifiers must be lower-case letters.

NOTE

In these and other examples, the elements of the escape sequences are separated by spaces. However, when you enter escape sequences at the terminal or print them from a program, all characters must be contiguous, as indicated below:

ESC&a5c10R

Parameters for escape sequences are enclosed in angle brackets (<>). Lastly, throughout this manual, 'ESC' and 'Ec' indicate the Escape Key.

Escape Sequence Programming

This section gives details and examples for escape sequence programming that will help you use the major features of the HP 150. The following topics appear in this section:

- o User-Definable Softkeys
- o Touchscreen Programming
- o Keyboard Control
- o Character Enhancements and Screen Control
- o Mode Selections
- o Block Transfers
- o Data Operations
- o Status Requests

In addition, Appendix D gives details and examples on Graphics Control Functions.

You will also find a summary of escape sequences in the *HP 150 Terminal User's Guide* which was included with your system.

INTRODUCTION TO SOFTKEYS

There are three sets of function keys on the HP 150: System Keys, User-Defined Keys, and Application Keys. You gain access to the system function keys through the [User System] key. These keys are maintained by the system. The other two sets of function keys can be programmed by your application. The first set is the "User-Definable Softkeys," which are programmed via escape sequences. The other set is the "Application Softkeys", which are controlled via AGIOS calls. (The section on AGIOS Programming contains more information about these calls.) These two sets of softkeys are summarized below.

User-Definable Softkeys

- o Are the set of 8 keys labeled [f1] through [f8] on the keyboard.
- o Are defined and displayed under program control by means of escape sequences.
- o Can also be displayed by pressing [CTRL] [User System] keys.
- o Can be programmed to perform terminal functions alone, or to simulate keyboard input with or without an automatic carriage return.
- o Can contain up to 80 characters in each key definition.
- o Can contain up to 16 characters on two lines in each key label, or can use the default label (e.g., [f1]).

Application Softkeys

- o Are the set of 8 keys labeled [f1] through [f8], together with the 4 unlabeled keys at the top of the numeric keypad on the keyboard.
- o Are defined, displayed, and controlled by AGIOS function calls.
- o Can also be displayed by pressing [Shift] [User System] keys.
- o Are normally used for input only in Keycode mode. The section on Keycode mode contains more information.
- o Return a keycode for each key, to be interpreted and processed by your application program.

For more information on Application Softkeys, refer to the section on Application Softkeys.

Defining the User Softkeys

The escape sequence that you use to program the User-Definable softkeys has the following general format:

```
ESC & f <attr> a <key> k <enh> v <lab half> x <lab len> d <buf len> L
      <lab> <buf>
```

As is true for all multi-character escape sequences, the HP 150 processes this sequence until the first upper-case character occurs. The characters in the above escape sequence are defined in the following way:

- ESC The escape character (ASCII 27 decimal).
- & f The "define function key" sequence.
- <attr> a The attribute. Values for <attr> are:
- 0 = Normal Key
 - 1 = Local Key
 - 2 = Transmit Key
- (See the note on the next page.)
- <key> k The softkey number. <key> must be in the range from 1 to 8, inclusive.
- <enh> v The enhancement for the label. Values for <enh> are:
- 0 None
 - 1 Blinking
 - 2 Inverse video
 - 3 Blinking and inverse video
 - 4 Underline
 - 5 Blinking and underline
 - 6 Inverse video and underline
 - 7 Blinking, inverse video, and underline
 - 8 Half-bright
 - 9 Blinking and half-bright
 - 10 Inverse video and half-bright
 - 11 Blinking, inverse video, and half-bright
 - 12 Underline and half-bright
 - 13 Blinking, underline, and half-bright
 - 14 Inverse video, underline, and half-bright
 - 15 Blinking, inverse video, underline, and half-bright
- <lab half> x The portion of the label to be enhanced. Values for <lab half> are:
- 1 Top half of label
 - 9 Bottom half of label

Escape Sequence Programming

<lab len> d The length of the softkey label; <lab len> must be between 0 and 80. However, only the first 16 characters will be used.

<buf len> L The length of the response buffer; <buf> must be between -1 and 80. A length of -1 clears the <buf>.

Notice that the character that follows <buf len> is an uppercase "L". This character ends the escape sequence. The <lab> and <buf> parameters that follow it are treated as data when the escape sequence is executed.

If you do not want to change the <buf len> parameter, make sure that the last parameter specified is an upper-case letter.

<lab> The ASCII characters to be displayed in the on-screen softkey labels.

<buf> The ASCII characters associated with the softkey.

For example, the following escape sequence defines the user-defined function key number 2 to be a Transmit key, softkey label is "Log on", softkey definition is "Hello manager.user", and use default settings for the other parameters:

```
ESC & f 2 k 2 a 6 d 16 LLog onHello Eunice.Yan
```

NOTE

Local defined keys execute only in the HP 150 display. None of the characters in <buf> are sent to the user program. Local keys generally are not useful in applications.

For Normal and Transmit keys, characters in <buf> are sent to standard console input. The principal difference is that in Transmit keys, a carriage return is appended to the end of each softkey buffer (hence transmitted to the internal computer). The characters in a Normal key buffer are sent to the program, but no carriage return is automatically appended. In order for your application to receive the characters, it has to use the correct handshaking. The section on Block Transfers has additional information on using handshake.

In BASIC, for example, the INPUT statement requires a carriage return to end input. If you are using the INPUT statement, be sure to define keys with the Transmit attribute.

However, the INPUT\$ function does not require a carriage return. If you are using this instruction for input, you can use Normal keys.

The sum of <lab len> and <buf len> must be less than 160 characters.

Displaying the User-Defined Softkeys

Once you define one (or more) of the User-Defined Softkeys, you will probably want to display them. To do so, use the following general escape sequence format, with the appropriate <parm> value:

ESC & j <parm>

The <parm> options and their meaning are listed below:

- @: This option turns off the labels currently displayed on the screen and the row and column indicators. It also turns off the Status Line, including the time and other status information.
- A: This option displays the Modes System Key labels on the screen. This is the level a user would see by pressing [User System] [f4] keys.
- B: This option displays the currently defined set of User-Defined Softkey labels. If no labels have been defined, the default labels [f1] through [f8] are used. This is the programmatic equivalent of pressing [CTRL][User System] keys.
- S: This options disables the [Menu] and [User System] keys, and locks the current softkey selections. Keys [f1] through [f8] can still be used.
- R: This option enables the [Menu] and [User System] keys.

When using the User-Defined Softkeys, you should keep in mind the following facts:

- o First, the User-Defined Softkeys remain "alive" even when the labels are not displayed. That is, when the User-Defined Softkeys are displayed using the ESC & j B sequence, and then turned off with the ESC & j @ sequence, the screen labels are still active and will return the softkey definition when touched or pressed.
- o Second, when you use the ESC & j sequence, the changed labels and definition string are not updated until you redisplay the labels by means of an ESC & j B sequence.
- o Finally, if Keycode mode is enabled, your application receives four bytes of input for each character in the <buf> field! See Keycode Mode in Section 4 for more information.

Displaying a User-Defined Message

There is one more valid parameter to the ESC & j sequence which requires slightly different syntax. The syntax of this sequence is:

ESC & j <len> L <message>

This sequence lets an application display up to 160 characters (two lines) of text in place of the softkey labels on rows 25 and 26 on your screen. The message is displayed on the screen until the [Return] key is pressed, and cannot be locked on the screen. The <len> parameter, which must be between 0 and 160 inclusive, specifies the number of bytes after the letter L that are to be displayed. The <message> buffer is displayed in place of the screen labeled softkeys.

Executing the User-Defined Softkeys

The following escape sequence executes the content currently assigned to the softkey specified in <x>:

ESC & f <x> E

X	Key
-----	-----
1	f1
2	f2
3	f3
4	f4
5	f5
6	f6
7	f7
8	f8

Programming Considerations

User-Defined Softkeys on the HP 150 are not interrupt-driven. This means that your application needs to check the keyboard buffer periodically for softkey presses.

Sometimes you will want softkey input to be indistinguishable from keyboard input. For example, an application menu may prompt the user for a numbered selection. By defining the softkey contents so that the selection number corresponds to the key number, the user can either type the menu selection or press the appropriate softkey. This procedure lets the User-Defined Softkeys, labeled appropriately, augment other keyboard input.

At other times, you may want softkey input to be unique. For example, an application may offer a user the ability to either type a filename or exit the program. If the user types any valid string, the application treats that input as a filename. If the 'exit' softkey is pressed, the application ends.

One way to solve this dilemma is to define the softkeys with control characters. Although the user can enter a control character from the console, your application can examine the character's ASCII value to distinguish between a normal ASCII input and the control-code definition of the softkeys.

To use this scheme successfully, you need to select softkey definitions that do not have unwanted effects on the operation of the system. ASCII values in the range from 18 through 25 do not interfere with system operation. Below this range, tab (8), line feed (10), carriage return (12), and DC1 (17) interfere. Above this range, the escape character (27) interferes. Given softkey definitions in the range from 18 through 25, here is a BASIC program segment that accepts keyboard input and sets a flag (KEY) to the softkey number when a softkey is pressed.

3000 A\$ = INPUT\$(1)	'Get character without echo
3010 A = ASC(A\$) : KEY = 0	'Get value; assume no KEY
3020 IF A<18 OR A>25 THEN 3040	'Not in softkey range
3030 KEY = A - 17	'Put KEY in range 1-8
3040 REM Continue with program	

USING TOUCHSCREEN

One unique feature of the HP 150 is the touchscreen. The touchscreen can be programmed in a variety of ways, giving you flexibility in designing your application. This section describes how to program the touchscreen and explains how you can make the best use of it in your application.

The first thing you need to know about programming the touchscreen is the "modes" of operation. The simplest and most flexible mode of operation involves "fields". In this mode, you tell the HP 150 information about where you want to detect touches, and the system firmware does most of the work.

As you will see, you can turn the touchscreen on and off without affecting the fields you have defined. You can select video enhancements for both "touched" and "untouched" fields, can control the positioning of the cursor, and can also tell the system to 'beep' when a field is touched.

A more comprehensive solution, but one which requires more work by the application, involves "row and column sensing". In this mode, you do not have to define any touch fields. The touchscreen passes absolute row and column addresses to your application. As with fields, you can turn sensing on and off, and can control cursor positioning and touch beeping.

Specifying Reporting Modes

The default mode for touchscreen reporting is OFF; that is, the HP 150 does not let your application know that the screen has been touched. Using the following escape sequence, you can enable, disable all touchscreen reports, and specify the form in which touches are reported.

```
ESC - z <smode> n [<tmode> m]
```

The meaning of each parameter is listed below:

- | | |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ESC | The escape character, ASCII 27 decimal. |
| - z | The sequence for setting the touch-sensing mode. |
| <smode> n | This parameter specifies the screen mode. Use this sequence to determine what type of reporting to perform. Possible values for <smode> are: |
| 0 | Turn off all reporting. This parameter turns off reporting without deleting any fields. Softkeys remain touch active unless you explicitly disable them using the Changing Softkey Touch Sensitivity sequence described later. |
| 1 | Enable row/column reporting only. This parameter disables field reporting, if active, and enables row/column reporting |

Escape Sequence Programming

only. You will receive row/column reports even when a field area is touched.

- 2 = Enable touch-field reporting only. The only reports you receive are from defined fields.
- 3 = Enable row/column and touch-field reporting. Touch fields are reported as defined. Row/column reports are made from all other areas of the screen.
- 4 = Toggle touchscreen ON/OFF. When the touchscreen is OFF, all touchscreen operations are disabled. This parameter also disables softkey fields. The start-up default value of this parameter is ON.

<tmode> M This parameter specifies the touch mode for row/column reporting; it determines when the HP 150 "tells" your application that touch is sensed. It is not required for touch fields because the sensing mode is set in the field definition. Moreover, this parameter is required for row/column sensing. If used with touch fields, this mode should correspond to the mode specified in the define-field escape sequence.

Valid parameters for <tmode> are:

- 1 = Report on touch only.
- 2 = Report on release only.
- 3 = Report on touch and release.

Defining a Touch Field

If you choose to use field reporting, the following escape sequence lets you define a touch field on the screen. Note that the "/" character is the continuation line marker, it is not part of the sequence.

```
ESC - z g <rows> r <cols> c < curs> p <beep> b /  
      <off_enh> e <on_enh> f <attr> a <mode> m /  
      <buf len> L <buf>
```

The meaning and valid entries for each of these fields are listed below.

ESC	The escape character, ASCII 27 decimal.
-zg	The sequence that represents a touch field definition.
<rows> r	Specifies the beginning and ending rows for this touch field. The <rows> parameter is specified as: <Start-row>,<End-row>

The valid ranges for Start-row and End-row are from 0 to 47 inclusive. End-row must be greater than or equal to the Start-row.

- <cols> c** Specifies the beginning and ending columns for this touch field. The **<cols>** parameter is specified as:
- <Start-col>,<End-col>**
- The valid ranges for Start-col and End-col are from 0 to 79 inclusive. End-col must be greater than or equal to the Start-col.
- <curs> p** This parameter specifies whether the alphanumeric cursor should be positioned at the upper left corner of the field when the field is touched. Possible values for **<curs>** are:
- 0 = The cursor does not move to the field.
- 1 = The cursor is positioned at the upper left corner of the field.
- If "p" is not specified, the cursor is not positioned on touch.
- <beep> b** The "b" parameter specifies whether the system should beep when the field is touched. Valid values for **<beep>** are listed below:
- 0 = No sound occurs when the field is touched.
- 1 = The system makes a beep when the field is touched.
- If "b" is not specified, no beeping occurs when the field is touched.
- <off_enh> e** This parameter specifies the enhancement (if any) which is displayed when the field is OFF (not touched). Possible values for **<off_enh>** are shown in the Touch Enhancement Table on this page.
- If "e" is not specified, the default off-enhancement of 10 is used. This displays a half-bright inverse field.
- <on_enh> f** This specifies the enhancement displayed when the field is ON, or touched. The possible values for **<on_enh>** are shown in the Touch Enhancements Table on this page.
- If "f" is not specified, the default on-enhancement of 2 is used. This causes an inverse video enhancement in the field.

Table 2-1. Touch Enhancements Table

Security OFF	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Security ON	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Half-bright										X	X	X	X	X	X	X
Underline					X	X	X	X						X	X	X
Inverse Video			X	X			X	X			X	X			X	X
Blinking		X		X		X		X		X		X		X		X

When security is ON, any data that is entered is not displayed. (Only the selected enhancement is displayed.)

Escape Sequence Programming

<attr> a This parameter specifies the type of field to be defined. The possible values for **<attr>** are listed below:

- 1 = ASCII field
- 2 = Keycode field (not supported with escape sequence)
- 3 = Toggle field
- 4 = Normal field

The next section entitled "Types of Touch Fields and Reporting" contains a full explanation of the types of fields.

<mode> m This parameter specifies the sensing mode for this field. The valid values for **<mode>** are listed below:

- 1 = Report on touch
- 2 = Report on release
- 3 = Report on touch and release

This parameter determines when the HP 150 "tells" your application that touch is sensed.

<buf len> L This parameter specifies the length of the response string associated with this field.

<buf> This parameter specifies the response buffer associated with this field. Although the parameter **<buf>** can be 0 to 80 bytes long for ASCII fields, it must be two characters in length for Toggle and Normal fields. These two characters will be reported back to your application in lower-case even though they may be defined as upper-case characters.

Types of Touch Fields and Reporting

The HP 150 lets you define the following four types of touch fields:

1. **ASCII FIELDS.** These fields are very similar to user-definable softkeys in that a buffer of up to 80 characters can be associated with every field. In an ASCII field, each time touch is sensed, your application receives the designated buffer as standard console input.
2. **KEYCODE FIELDS.** This type of field lets you specify a Qualifier word and a Data word. When this field is touched, the above two words are returned to your application. These fields require Keycode mode, and is done via assembly language programming. To define a keycode touch field, you need to use the AGIOS function (0,32). The subsection entitled "Touchscreen Functions" contains more information about this function.
3. **TOGGLE FIELDS.** Toggle fields are "regions" for which ON and OFF states make sense. When the field is first touched it is toggled on. When the field is touched again, it is toggled off. The following escape sequence is returned to your application according to the sensing mode parameter you specified.

ESC - z <buf> <type> Q

<Buf> is the two-character buffer specified when the toggle field is defined. You should note that the characters are always returned in lower case. This two-character buffer could be used to identify the touch field. The <type> parameter will return the following values:

- 1 = Toggle field turning on
- 2 = Toggle field turning off

4. NORMAL FIELDS. These fields are similar to ASCII fields, but you use only a two-character <buf> parameter, as you do with toggle fields. The characters are always returned in lower case. The following escape sequence is returned for normal fields:.

ESC - z <buf> <type> Q

The <buf> parameter is the two-character buffer specified when the field is defined. Valid return values for <type> are:

- 5 = Normal field touch sensed
- 6 = Normal field release sensed

The main difference between Toggle Fields and Normal Fields is that Toggle Fields have two states, (ON and OFF). A Normal Field is ON only while it is actually being touched.

Row/Column Reporting

The alternative to touch field reporting is row/column reporting. When row/column reporting is in effect, the touch reports that you receive have the following format:

ESC - z <row> x <col> y <type> Q

The <row> and <col> parameters are the two-digit integers that specify the row and column positions of the touch. <row> is from 0 through 47 and <col> is from 0 through 79. Possible values for <type> are:

- 3 = Row/Column touch reporting
- 4 = Row/Column release reporting

When you use row/column reporting, you do not have to define any touch fields. The <tmode> parameter in the Specifying Reporting Mode escape sequence lets you specify the touch mode for row/column reporting.

Deleting Touch Fields

You may want to delete all touch fields or remove one or more selected fields. For example, if several options are presented as touch fields, you may want to remove all of the fields and continue to another menu. Or else, you may want to invalidate only certain fields, by deleting the fields that you do not want to use.

The following escape sequence lets you delete a touch field other than a softkey field. If no field is defined at the specified coordinates, no action is taken.

```
ESC - z d <row> r <col> C
```

NOTE

Any coordinates within a defined touch field will delete the entire field.

The second form of this escape sequence, given below, deletes all touch fields. You should use this sequence whenever your application moves from one menu of touch fields to another menu of touch fields.

```
ESC - z D
```

Whenever a touch field is defined, the touch-field information is stored in terminal memory. Therefore, deleting fields conserves terminal memory.

Changing Softkey Touch Sensitivity

The User-Definable Softkeys are touch sensitive by default. (The section on User-Definable Softkeys explains how to define these keys.) If you want to change the touch-sensitive nature of the softkeys, use the following escape sequence:

ESC - z <key> s <mode> K

The parameters are listed below:

ESC	The escape character, ASCII 27 decimal.
-z	The sequence indicating touchscreen control.
<key> s	The softkey number. <key> can be from 1 through 8 inclusive.
<mode> K	The key mode. Its values are: 0 = Disable touch on <key> 1 = Enable touch on <key>

Resetting the Touchscreen

The following escape sequence resets all touch fields.

ESC - z J

Resetting the touchscreen turns all fields to the OFF state. This action affects the field state, but does not affect touch-sensing or reporting.

Programming Considerations

The touchscreen provides a very easy way to add unique features to your application. However, there are a few things to keep in mind when you are using the touchscreen.

You probably want to select one type of field and stick with it. ASCII fields are probably the best, because it is easy to equate them with valid keyboard input. This way your user can specify input either through the keyboard or via the touchscreen. If you use another type of field, you will have to parse the escape sequence that is returned to your application in order to determine which touch field was affected (touched) and how it was affected.

You should also keep touch field responses to a minimum number of bytes, for the following two reasons:

1. Your application screens will experience a minimal interference from touch input.
2. You conserve the limited terminal memory space. Every time you create a touch field, additional memory is allocated to store the field information. This happens even if a defined field already exists in the same location because touch fields can overlay each other.

This second point leads to another point to remember. When you are finished with a touch field, delete it! Lastly, you should not re-define a field over and over in a loop. The following two program segments illustrate this point:

PROGRAM A

```
1000 PRINT HOMEUP$;      'Clear screen; position cursor at top
1010 GOSUB 2000           'Define a touch field (actual
                          ' field not critical)
1020 A$=INPUT$(1)        'Read one character from touchscreen
                          ' or from keyboard
1030 IF A$="1" GOTO 1500  'Go off on option 1
1040 GOTO 1010           'Not option 1, return for more
```

Now look at Program B:

PROGRAM B

```
1000 PRINT HOMEUP$;      'Clear screen; position cursor at top
1010 GOSUB 2000           'Define a touch field (actual
                          ' field not critical)
1020 A$=INPUT$(1)        'Read one character from touchscreen
                          ' or from keyboard
1030 IF A$="1" GOTO 1500  'Go off on option 1
1040 GOTO 1020           'Not option 1, return for more
```

Escape Sequence Programming

See the difference? Look at line 1040. In Program A, the touch field is defined each time a character is accepted. In Program B, the touch field is defined only once. Program A eventually uses up the available memory in the system.

When you use touchscreen, the touch fields "auto-repeats", just as the keys on the keyboard do. To prevent this action from complicating your application, use the touch-sensing-mode (ESC - z <smode> N) escape sequence to control acceptance of input. For example, notice how the following program controls the touchscreen:

PROGRAM C

```
1000 PRINT HOMEUP$;          'Clear screen; position cursor at top
1010 GOSUB 2000               'Define a touch field
1015 PRINT CHR$(27);"-z2N";   'Turn on touch-field sensing
1020 A$=INPUT$(1)             'Read one character from touchscreen
                                ' or from keyboard
1025 PRINT CHR$(27);"-zON";   'Turn off sensing
1030 IF A$="1" GOTO 1500      'Go off on option 1
1040 GOTO 1015                'Not option 1, return for more
. . .
2000 PRINT CHR$(27);"-zgl,5rl,5clbla2mlL1;
2005 RETURN
```

Note lines 1015 and 1025. They turn on sensing just before input is expected, and turn off sensing immediately after input is received.

Of course, at line 1500 in all of the above examples, you should delete the selected touch field to avoid inputting additional (unwanted) characters. (*One possible exception might be the case in which program control returns to line 1020.*)

If more than one field is defined over a particular area of the screen, the HP 150 reports only the *most recently* defined buffer. This fact can prove useful, because sometimes you will want to know whether the user is touching the screen but is not touching a defined field. You can obtain this information by defining the entire screen as a field with no enhancements for ON or OFF states. Then define your application fields over this background field. When the user touches one of your fields, you receive the buffer you expect. But if the user touches any other part of the screen, you receive the buffer associated with the background field.

CONTROLLING THE KEYBOARD

Some applications on the HP 150 need more control over the keyboard than is required for data entry alone. Some of the functions that you may want to trap in your application are listed below:

- o Cursor Control Keys
- o [Next] and [Prev] Keys
- o Scroll Up and Scroll Down Keys
- o Character and Line Manipulation Keys

In general, keyboard functions that can be executed by escape sequences can be trapped. To trap keyboard functions, you should enable Transmit Function Key mode. This mode causes the terminal to transmit the escape sequence associated with the function, instead of executing the function normally.

Transmit Function Key Mode

You control the Transmit Function Key mode by setting or clearing the "A" strap in the HP 150 Terminal Configuration Menu. The user can do so (see the *HP 150 Personal Computer Owner's Guide* for details) or else you can use the following escape sequence:

ESC & s 1 A Enable Transmit Function Key mode

ESC & s 0 A Disable Transmit Function Key mode

For example, if you press [Clear line], in normal operation, the terminal erases all of the characters that appear between the current cursor position and the end of the line. In Transmit Function Key mode, the terminal sends two characters (ESC K) to your application. The HP 150 does not erase data to the end of the line unless your application echos or prints the two characters to the display.

The two-character escape sequence (ESC K) returned to your application is the same sequence that clears a line. The characters returned to your application in Transmit Function Key mode are always the escape sequence characters that correspond to the function of the key.

The terminal executes the function only when the function is echoed to the console. Therefore, your application should use a console input statement that does not echo to the console. The following BASIC program segment contains an example of such an input statement. In this segment, once Transmit Function Key mode is enabled, the only control key that is executed is the [Clear line] key, which generates ESC K.

1000 PRINT CHR\$(27); "&k1A"	'Enable Transmit Function Key mode
2000 CH\$ = INPUT\$(1)	'Read one character without echo
2010 IF CH\$ <> ESC\$ THEN 2070	'Not ESC character
2020 CH\$ = INPUT\$(1)	'Read the next character
2030 IF CH\$ = "K" THEN 2050	'Clear Line
2040 GO TO 2000	
2050 PRINT ESC\$;"K";	
2060 GO TO 2000	'Go get another character
2070 PRINT CH\$;	'Not an escape sequence, so echo
	' the key!

Programming Considerations

Transmit Function Key mode gives you additional control over the HP 150. When using this mode, keep the following points in mind.

First, if you set Transmit Function Key mode ON when your application starts, be sure to turn it OFF when you exit your application. This action puts the HP 150 in its default state, and ensures that your application does not disrupt other applications.

The Transmit Function Key mode causes the terminal to send an escape sequence to your application. The sequence you receive is the same sequence which, if sent by your application, would perform the same task as the key. Thus, to control the keyboard properly, your application should perform input without echo. This way, your application receives the associated escape sequence and can determine a suitable course of action before the HP 150 performs the task.

You should also remember that the default value of each user-defined softkey returns a two-character escape sequence. These default values are listed below:

[f1]	ESC p
[f2]	ESC q
[f3]	ESC r
[f4]	ESC s
[f5]	ESC t
[f6]	ESC u
[f7]	ESC v
[f8]	ESC w

Although you can use these default values, you may find it easier to define the softkeys to fit your application.

Finally, some keyboard controls do not generate escape sequences. For these, you will need to use AGIOS. With AGIOS, you can specify all of the keyboard control keys or 'special' keys to execute normally, to be intercepted by your application, or to be ignored. If you need more control than is available via the Transmit Function Key mode, you will need to incorporate AGIOS functions into your application. For more information, see "Keycode Mode" and "Keyboard Intercept" in the section on Alphanumeric Input/Output Subsystem Function Reference.

CONTROLLING THE DISPLAY

The display enhancements that the HP 150 supports include video enhancements and alternate character sets. You can also 'turn off' the alphanumeric display, either with or without having this action affect the softkey labels.

Display Enhancements

Four types of display enhancements are available on the HP 150: Blinking, Inverse, Underline, and Half-Bright. You can use these enhancements one at a time or in any combination.

The general format of the escape sequence that enables enhanced text is:

ESC & d <enh>

This escape sequence lets you specify that the display enhancement indicated by <enh> will be in effect from the present cursor position to the end of the current line. The following table lists the value for <enh>:

Table 2-2. Display Enhancements Table

<enh>	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	S
Blinking		*		*		*		*		*		*		*		*	
Inverse			*	*			*	*			*	*			*	*	
Underline					*	*	*	*					*	*	*	*	
Half-Bright									*	*	*	*	*	*	*	*	
No Enhancement	*																
Security																	*

Security as an enhancement lets you define an area in which text will not be echoed to your screen. This type of enhancement is useful when you do not want the actual text to display on the screen, such as when entering a password.

Once enabled, an enhancement remains ON until it is turned off (via an escape sequence in which <enh> is '@'), or until the cursor is moved to a different line.

Selecting a Character Set

The HP 150 includes several character sets that you can include in your application. These sets are described below:

<u>Character Set</u>	<u>Description</u>
Base	The system's base set. This is normally the U.S. ASCII character set.
Math	This is a set of mathematics characters, indicating Greek letters and various symbols.
Line	The Line-Drawing set lets you use various line segments, corners, and symbols.

Two other character sets can be accessed only through AGIOS functions: an *Italics* character set and a Bold character set. "Video Intrinsic" in Section 4 contains more information about using those character sets.

The escape sequence for selecting an alternate character set is:

ESC) <cset>

Values for <cset>, the character set selection, are listed below:

@ = Base set selected as alternate character set
A = Math set selected as alternate character set
B = Line-Drawing set selected as alternate character set

Once you have selected one of these sets as the alternate set, you can shift between the base set and the alternate set by means of the ASCII Shift-Out (SO) and Shift-In (SI) characters. The decimal values for these characters are 14 and 15, respectively. From the keyboard, you shift out to an alternate set by simultaneously pressing the [CTRL] and "N" keys. You shift in to the base set by simultaneously pressing the [CTRL] and "O" keys.

Escape Sequence Programming

You can use the alternate set by printing the ASCII character that corresponds to the desired character, as shown in the following table:

Table 2-3. Alternate Character Sets

<cset>	Character Set Equivalents
--------	---------------------------

Line Drawing Set

! 1	@ 2	# 3	\$ 4	% 5	^ 6	& 7	* 8	(9) 0	[-	+ =	←	~
Q	W	E	R	T	Y	U	I	O	P	{	}	--	!
L	S	D	F	G	H	J	K	M	L	:	"	†	
-	Z	X	C	V	B	N	M	+	+	+	+	+	+

Math Set

√ 1 1	!@ 2 2	# 3 3	\$% 4 4	± 5 5	^& 6 6	* 7 7	+ () 8 8	= 9 9	0 0	[-] =	+ = =	← ←	~ ~
γ _q	Q _w	E _θ	R _r	T _τ	Y _υ	U _ξ	I _ι	O _ρ	P _π	{ _↑	} _↑	→ _↓	→ _↓
α _A	S _σ	Φ _d	F _δ	G _λ	H _η	Θ _j	K _κ	ω _l	Λ _Δ	! _;	" _;		
ζ _Z	χ _x	Ψ _c	Δ _v	β _b	ν _n	μ _m	Ω _ψ	< _†	> _‡	Σ _?	≡ _/		

You can use this table to write programs that draw boxes and lines, that use special math symbols, and that even enhance these special characters. Here is a sample BASIC program that draws a rectangle on the screen:

```

2500 'Print a box with 'Hello There" in the middle
2510 PRINT CHR$(27);")B";      'Select Line Drawing as Alternate"
2520 PRINT CHR$(14);"Q;;;;;;;;;;;;;;;"W" 'Top line
2525 ' Now print middle line with "normal" characters
2530 PRINT CHR$(14);".";CHR$(15);" Hello There ";
2540 PRINT CHR$(14);"." 'Other end of box
2545 ' That concludes the middle line
2550 PRINT CHR$(14);"A;;;;;;;;;;;;;;;"S" 'Bottom of box
    
```

Note that to print the "normal" characters inside the box, you must shift back to the primary character set (lines 2525 through 2545).

This program produces the following figure:

Hello There

Alphanumeric Memory Control

The alphanumeric screen and the graphics screen are independent. Therefore, both of these screens can be displayed together, or either can be displayed alone. If Graphics Text mode is ON, then alphanumeric characters are written into graphics memory. This mode is controlled totally within graphics operations.

In the alphanumeric memory, you can turn off the entire display (including the softkeys); or else you can turn off any text and leave the softkey labels ON. To perform these tasks, use the following escape sequences:

ESC & w 12 F	Alpha Display ON
ESC & w 13 F	Alpha Display OFF (softkey labels remain)
ESC * d E	Alpha Memory ON
ESC * d F	Alpha Memory OFF (including softkey labels)

Appendix D, entitled "Graphics Control Functions," contains more information about the latter two sequences (ESC * d E and ESC * d F).

Programming Considerations

When you use display enhancements and alternate character sets, keep in mind how the HP 150 works. When the display is cleared, none of alpha memory is allocated. When you position the cursor and turn on a display enhancement, that enhancement affects all bytes on that line that are allocated now or that may be allocated in the future.

For example, position the cursor in column 5 on the first row. Invoke the 'inverse video' enhancement by using the escape sequence ESC & d B. Now print several characters; they will appear in inverse video. If your application leaves the first line and then returns to column 70 on the first line, the next characters you print will cause the entire line to appear in inverse video. This change occurs *even if you do not include the 'inverse video' sequence at column 70!*

To avoid this situation, you should turn on inverse video; print the text that you want to appear in inverse; and turn off inverse video (by using the ESC & d @ sequence). Thus, repositioning the cursor to the right on the same line will not cause the entire line to be enhanced.

Alternate characters work the same way. Once you have shifted out (SO) to an alternate set, all characters to the right will also be shifted, unless and until you print a shift-in character (SI).

USING THE KEYBOARD AND THE DISPLAY

When the following two-character escape sequences are executed from a program, they perform functions that you can also perform from the keyboard. The *HP 150 Terminal User's Guide* contains additional information about these escape sequences.

Two-Character Escape Sequence

Table 2-4. Two-Character Escape Sequences

<u>Escape Sequence</u>	<u>Function</u>
ESC 0	Copy information in display memory to destination(s)
ESC 1	Set tab
ESC 2	Clear tab
ESC 3	Clear all tabs
ESC 4	Set left margin
ESC 5	Set right margin
ESC 6	Define alphabetic-only field to be used by Format mode
ESC 7	Define numeric-only field to be used by Format mode
ESC 8	Define unrestricted field (alphabetic or numeric characters) to be used by Format mode
ESC 9	Clear all margins
ESC @	Delay one second
ESC A	Move cursor up one line
ESC B	Move cursor down one line
ESC C	Move cursor right one space
ESC D	Move cursor left one space
ESC E	Hard reset (power-on reset)
ESC F	Home cursor down (to lower left corner of display)

Escape Sequence Programming

<u>Escape Sequence</u>	<u>Function</u>
ESC G	Move cursor to left margin
ESC H	Home cursor up (to upper left corner of display)
ESC I	Horizontal tab
ESC J	Clear display from cursor to end of display memory
ESC K	Clear line from cursor to end of line
ESC L	Insert one blank line
ESC M	Delete one line
ESC N	Turn on Insert Character with wraparound mode
ESC O	Delete one character at current cursor position with wraparound
ESC P	Delete one character at current cursor position without wraparound
ESC Q	Turn on Insert Character mode (insert character without wraparound)
ESC R	End Insert Character mode
ESC U	Next Page
ESC V	Previous Page
ESC W	Format mode ON
ESC X	Format mode OFF
ESC Y	Display Functions mode ON
ESC Z	Display Function mode OFF
ESC [Start unprotected field
ESC]	End unprotected/transmit-only field
ESC ^	Primary terminal-status request
ESC _	Write non-displaying terminator
ESC `	Sense cursor position (relative)

<u>Escape Sequence</u>	<u>Function</u>
ESC a	Sense cursor position (absolute)
ESC b	Unlock keyboard
ESC c	Lock keyboard
ESC d	Transmit a block of text to computer (simulate the ENTER key)
ESC f	Disconnect Modem
ESC g	Soft reset
ESC h	Home cursor up to the first unprotected field (ignoring transmit-only fields)
ESC i	Backtab
ESC j	Display User-Definable Function key menu and begin User Key Definition mode
ESC k	Restore normal display and end User Key Definition mode
ESC l	Begin Memory Lock mode
ESC m	End Memory Lock mode
ESC p	Default definition for user-definable function key f1
ESC q	Default definition for user-definable function key f2
ESC r	Default Definition for user-definable function key f3
ESC s	Default Definition for user-definable function key f4
ESC t	Default Definition for user-definable function key f5
ESC u	Default Definition for user-definable function key f6
ESC v	Default Definition for user-definable function key f7
ESC w	Default Definition for user-definable function key f8
ESC x	Initiate datacomm self-test
ESC z	Initiate terminal self-test

Escape Sequence Programming

<u>Escape Sequence</u>	<u>Function</u>
ESC {	Start transmit-only field
ESC	Erase non-displaying terminator (for use in Format mode)
ESC -	Secondary terminal-status request

Display Structure

The display portion of the terminal consists of the display memory and the display screen. Display memory is the portion of terminal memory that contains alphanumeric data entered to the terminal or displayed on the screen. Display memory consists of 48 rows (numbered 0 through 47) and 80 columns (numbered 0 through 79).

The display screen, which is the visible part of display memory, consists of 27 rows. Each row contains space for 80 characters (numbered 0 through 79). The first 24 rows of the display screen shows one "page" of display memory. Rows 25 and 26 display the function-key labels ("softkeys"). Row 27 contains information about the terminal's status, and also shows the current date and time.

As data is added to display memory, all of display memory will eventually be used. When display memory is full, lines are rolled off the top of the display and they are lost. The Status line indicates the number of lines entered into display memory.

The cursor, which appears on the screen as a blinking underscore mark or a blinking box in inverse video, indicates where the next character entered will appear. The cursor appears only on the display screen.

Memory Addressing

Display memory can be addressed using absolute or relative coordinate values. The three types of addressing are listed below:

- o Absolute
- o Screen-Relative
- o Cursor-Relative

In absolute addressing, you specify the row and column in display memory. Column values range from 0 through 79, and row values range from 0 through 47.

In screen-relative addressing, you specify the position relative to the 'window' where the first line on the screen is row 0. If scrolling has occurred, for example, screen relative row 0 may be absolute row 10.

In cursor-relative addressing, you specify a position relative to the current cursor position by specifying signed row and column values. A plus sign ("+") indicates movement to the right or down; a minus sign ("-") indicates movement to the left or up.

If the row or column addresses are greater than the available addresses, the largest possible value is used. In screen-relative addressing, the cursor cannot be moved to a row position that is not currently displayed.

Escape Sequence Programming

The cursor control operations are described below:

<u>Escape Sequences</u>	<u>Function</u>
ESC & a <col> c <row> R	Moves the cursor to column <col> and row <row> in display memory (Absolute addressing)
ESC & a <col> c <row> Y	Moves the cursor to column <col> and row <row> in screen memory (screen relative addressing).
ESC & a <scol> c <srow> Y	Moves the cursor to column <scol> and row <srow> on the screen relative to its present position. The <scol> and <srow> parameters represent signed numeric ASCII characters. A positive number indicates movement up or to the right; a negative number indicates movement down or to the left.
ESC & a <scol> c <srow> R	Moves the cursor to column <scol> and row <srow> relative to its present position in memory.

You can use a combination of display-screen, display-memory, and cursor-relative coordinates in one escape sequence.

Examples:

1. The following escape sequence positions the cursor at column 70, 18 rows below the current cursor location. (If necessary, text will scroll up so that the cursor can be moved to the specified line.)

ESC & a 69 c + 18 R
2. The following escape sequence positions the cursor 15 columns to the left of the current cursor position, in the 4th row currently visible on the display screen.

ESC & a -15 c 3 Y
3. The following escape sequence positions the cursor at column 5 on the current row:

ESC & a 4 C

Programming Considerations

In cursor positioning, you should be aware of the effect of memory lock. When memory lock is enabled, 'freezing' part of the text at the top of the screen, absolute memory addressing does not function properly. For safety's sake, always use screen-relative addressing.

If you do not want to scroll, you can effectively disable it by using the Transmit Function Key mode in combination with 'home up' commands whenever you paint a form on the screen. This technique lets you capture the scrolling functions, and ensures that your addressing is always correct. Transmit Function Key mode is explained earlier in the section on Controlling the Keyboard.

MODE SELECTIONS

You can use escape sequences to change the active values of the mode function keys and terminal configuration parameters listed in Table 2-5 below. A change in a parameter value made through one of the sequences listed below takes effect immediately. However, the values in non-volatile memory are not changed; that is, the default values are restored when you restart your operating system. If a configuration menu is displayed on the screen when the terminal receives the escape sequence, the sequence is not executed until you exit from the menu.

You can use an escape sequence to lock the current configuration menus so that they cannot be altered from the keyboard or from a program. Any attempt to gain access to a locked menu causes a "beep" from the bell, and generates an error message. When the configuration menus are locked, the MODIFY ALL, BLOCK MODE, REMOTE MODE, and AUTO LF function keys are also locked.

To lock the menus, use the following escape sequence:

```
ESC &q 1L
```

To unlock the menus, use the following escape sequence:

```
ESC &q 0L
```

Mode Selection Escape Sequences

The following escape sequences select active values without changing the values in non-volatile memory. Entries in the MENU FIELD/MODE column that are marked with an asterisk appear on the Terminal Configuration Menu. These fields are defined in the Function Keys chapter in the *HP 150 Terminal User's Guide*.

Table 2-5. Mode Selection

ESCAPE SEQUENCE	MENU FIELD/ MODE	VALUES	VALUE OF x
ESC &k <x>A	Auto Lf	OFF	x=0
		ON	x=1
ESC &k <x>B	Block Mode	OFF	x=0
		ON	x=1
ESC &k <x>C	*Caps Lock	OFF	x=0
		ON	x=1
ESC &k <x>D	*Bell	OFF	x=0
		ON	x=1
ESC &k <x>I	*ASCII 8 Bits	NO	x=0
		YES	x=1
ESC &k <x>K	Auto Keyboard Lock mode	OFF	x=0
		ON	x=1
ESC &k <x>L	*Local Echo	OFF	x=0
		ON	x=1
ESC &k <x>M	Modify All	OFF	x=0
		ON	x=1
ESC &k <x>N	SPOW(B)	OFF	x=0
		ON	x=1
ESC &k <x>O	Numeric/ Graphics pad	Numeric	x=0
		Graphics	x=1
ESC &k <x>P	Caps mode	OFF	x=0
		ON	x=1
ESC &k <x>Q	Click	OFF	x=0
		ON	x=1
ESC &k <x>R	Remote Mode	OFF	x=0
		ON	x=1
ESC &s <x>A	*XmitFnctn(A)	NO	x=0
		YES	x=1
ESC &s <x>B	*SPOW(B)	NO	x=0
		YES	x=1
ESC &s <x>C	*InhEolWrp(C)	NO	x=0
		YES	x=1

Escape Sequence Programming

ESCAPE SEQUENCE	MENU FIELD/ MODE	VALUES	VALUE OF x
ESC &s <x>D	*Line/Page(D)	LINE PAGE	x=0 x=1
ESC &s <x>G	*InhHndShk(G)	NO YES	x=0 x=1
ESC &s <x>H	*Inh DC2(H)	NO YES	x=0 x=1
ESC &s <x>J	*Auto Term(J)	NO YES	x=0 x=1
ESC &s <x>K	*ClearTerm(K)	NO YES	x=0 x=1
ESC &s <x>L	*InhSlfTst(L)	NO YES	x=0 x=1
ESC &s <x>N	*ESC Xfer(N)	NO YES	x=0 x=1
ESC &s <x>W	*InhDcTst(W)	NO YES	x=0 x=1
ESC &x <x>C	Send Cursor Position	OFF ON	x=0 x=1

Programming Considerations

When your application first executes, you should send a 'set-up string' to the HP 150 to initialize the state of all the straps and modes.

The following sequence is a good starting point for all applications:

```
ESC & k 0 a   Auto Line Feed OFF
        0 b   Block mode OFF
        0 c   Caps Lock OFF
        1 d   Bell ON
        0 i   ASCII 7 bit mode
        0 k   Keyboard Lock OFF
        0 l   Local Echo OFF (MS-DOS echoes character)
        0 m   Modify All mode OFF
        0 n   Space Overwrite mode OFF
        0 o   Numeric Keypad ON
        0 p   Caps mode OFF
        1 Q   Keyclick ON
```

You should also consider using the following escape sequences:

```
ESC & s 0 a   Transmit Function Keys OFF*
        0 b   Space Overwrite disabled
        0 c   Inhibit end-of-line wraparound
        0 d   Line mode enable
        1 g   Inhibit Block mode handshaking
        1 h   Inhibit DC2 Trigger
        0 j   Auto Terminator disabled
        0 k   Clear Terminator mode OFF
        1 l   Inhibit Self-Test
        0 n   Escape sequence transmit to printer disabled
        1 W   Inhibit DataComm Test
```

*If you want to use the Transmit Function Keys mode, this parameter will be 1 a.

FORMAT MODE

Format mode lets your application transmit data from unprotected fields on the screen to a host computer. You define unprotected fields when Format mode is disabled. When Format mode is enabled, you can enter and transmit data. (See the *HP 150 Terminal User's Guide* for additional information on defining fields.)

Defining Fields

From an application, you can use the following escape sequences to define "unprotected" and "transmit-only" fields with the various screen attributes:

ESC [Starts an unprotected field

ESC { Starts a transmit-only field

ESC] Ends a field

ESC & k <x> Z The data is transmitted when the ENTER key is pressed, according to the following values for <x>:

- 0 = Transmits data within the unprotected and transmit-only fields (default)
- 1 = Transmits data within transmit-only fields, and within any unprotected fields that have been modified.

The two-character escape sequences ESC 6, ESC 7, and ESC 8 define various attributes of each field or subfield. The meanings of these sequences are listed below:

ESC 6 Specifies an alphabetic field only. No numerical or special characters are accepted.

ESC 7 Specifies a numerical only field. No alphabetical or special characters are accepted.

ESC 8 Specifies an unrestricted field. Any character (alphabetical, numerical, or special) can be entered within this field.

The same sequence of operations applies when fields and subfields are defined programmatically as when they are defined through the keyboard. For example, if you want a field to include video enhancements, you must send the appropriate ESC & d sequence before sending ESC [or ESC { sequence. To define the start of a subfield, you must first send the appropriate sequence (ESC 6, ESC 7, or ESC 8) at the point where the subfield is to begin.

Forms are defined when Format mode is disabled. However, forms are not interpreted as such until Format mode is enabled. When Format mode is initiated, all of the display memory is "protected", except for the portions that have been explicitly defined as "unprotected" and "transmit-only" fields.

You can use the following escape sequences to enable and disable Format mode programmatically:

```
Enable Format mode : ESC W
Disable Format mode : ESC X
```

Transferring Forms from the Screen to a Host Computer

When writing application programs that will display a form on the terminal screen, you may decide to code the program statements that issue the necessary escape sequences, shift-out (SO) and shift-in (SI) codes, and data. If the form structure is complex, this method can be both tedious and prone to error.

An easier way is to design the form at the terminal when it is not connected to either a remote host computer or MS-DOS. And then transfer the form structure from the screen to the host computer, where it can be accessed by or incorporated into your program. If the terminal is connected to an HP 3000 Computer System, you can use the FORMSPEC portion of V/3000, and then include appropriate V/3000 intrinsic calls in your application programs to use the form while an application program is executing.

BLOCK TRANSFERS

The HP 150 normally operates in Character mode, sending each character to the host as the key is pressed. The HP 150 can also operate in Block mode, in which data is sent to the host one block at a time. Block mode requires special handshaking with the host. (As used here, 'Host' indicates a remote system or the MS-DOS operating system.)

Even in Character mode, certain transfers occur in Block mode (all of which occur in Remote mode).

- o A transfer is initiated when the Enter key is pressed.
- o A transfer is initiated by the escape sequence ESC d (the Enter key sequence).
- o Data is transferred when Line Modify or Modify All mode is ON.
- o The definition string of a "Transmit" type user-defined softkey is transferred when the key is pressed in Remote mode.
- o Status-data transfers (such as information about the cursor position or about the terminal status) are requested.

The actual protocol of the data transfer depends on how the following configuration switches are set.

<u>Terminal Parameter:</u>	<u>Default Setting:</u>
Line/Page (D)	Line
InHndShk (G)	No
Inh DC2 (H)	No
AutoTerm (J)	No
ClearTerm (K)	No

Block/Character, Line Modify/Modify All, Format, and Auto Line Feed modes also affect data transfer. The following table shows the configuration menu settings that let you select each mode.

Table 2-6. Data Transfer Settings

TRANSFER TYPE	HANDSHAKE TYPE	MODE	G STRAP	H STRAP
Enter key	None	Character	No	-
	None	-	-	Yes
	DC1	(Option not available)		
	DC1/DC2/DC1	Block	-	No
	DC1/DC2/DC1	-	Yes	No
Status and ESC d	None	-	Yes	Yes
	DC1	-	No	-
	DC1/DC2/DC1	-	Yes	No
User-defined function key (Transmit)	None	Block Page	-	Yes
	None	-	Yes	Yes
	DC1	Block Line	No	-
	DC1	Character	No	-
	DC1/DC2/DC1	-	Yes	No
	DC1/DC2/DC1	Block Page	-	No
Modify modes	None	-	-	Yes
	None	-	No	No
	DC1	(Option not available)		
	DC1/DC2/DC1	-	Yes	No

NOTE

A "-" symbol in the above table indicates a "don't care".

Escape Sequence Programming

To accept a Block mode transfer programmatically, you must perform the following steps:

1. Disable Auto Line Feed
2. Use the correct form of handshaking
3. Decide on the most appropriate way to receive data

These steps are explained in the following paragraphs.

Handshaking

In general, the `InhHndShk(G)` (Inhibit Handshake, strap G), and `InhDC2(H)` (Inhibit DC2, strap H) selections on the Terminal Configuration Menu determine the type of transfer handshaking to be used when transferring blocks of data from the terminal to the host computer. You can use the `ESC & s` sequence to set the G and H straps.

The following three handshakes are possible:

1. No handshake (Type 1), in which the terminal simply sends the data block without any special control from the host system. This handshake occurs when both the 'G' and 'H' straps are set.

`InhHndShk (G) = Yes`
`Inh DC2 (H) = Yes` or `ESC & s 1 g 1 H`

2. A DC1 Trigger handshake (Type 2), in which the host computer must trigger the block transfer with a DC1 character. This handshake occurs when the 'G' strap is not set and the 'H' strap is ignored.

`InhHndShk (G) = No`
`Inh DC2 (H) = Yes or No` or `ESC & s 0 g 0 H`

3. A DC1/DC2/DC1 handshake (Type 3), in which a DC1 from the host is answered by a DC2 from the terminal. The host must respond with a second DC1. When this response occurs, the terminal transmits the data. This handshake occurs when the 'G' strap is not set and the 'H' strap is set.

`InhHndShk (G) = Yes`
`Inh DC2 (H) = No` or `ESC & s 1 g 0 H`

Correct Handshaking

The Microsoft languages do not output a DC1 as a signal that they are ready to receive data. If the terminal configuration is set for the default handshaking scheme (DC1/DC2), and if a program outputs an escape sequence asking for a Block mode transmission, the terminal will lock up while it waits for the DC1 trigger. At this point, you will see the message "KB Locked" in the lower left corner of your screen. To eliminate this problem, many users disable handshaking entirely. Although this tactic works in most cases, it can cause data overruns. Another disadvantage is that the user must re-enable handshaking each time he or she wants to access a host system.

A more conservative approach is to leave DC1/DC2 handshaking enabled and have the program output the DC1 trigger. The following BASIC program segment illustrates this procedure:

```
1000 REM Read one line from the display
1010 PRINT CHR$(27);"d";CHR$(17);
1020 LINE INPUT TEXT$
```

The PRINT statement outputs an ESC d to request the transfer of one line from the display, and then outputs a DC1 to inform the terminal that the program is ready to receive.

With regard to compatibility, a program that does not output its own DC1s must be run on a system on which handshaking is disabled. If the program does output its own DC1, it will run regardless of whether handshaking is disabled. If the DC1/DC2 handshake is disabled, and if the terminal receives the DC1 sent by your program, the terminal will simply ignore it.

Disable Auto Line Feed

When Auto Line Feed is enabled, all block transfers are preceded by a line-feed character. This condition can cause misalignment of the cursor before the transfer begins; it can also cause difficulties in reading the data transmitted. Therefore, we recommend that you disable Auto Line Feed.

You can use the following escape sequence to disable Auto Line Feed:

```
ESC & k 0 A
```

The Appropriate Way to Receive Data

The most common block-mode transmissions are generated by one of the three escape-sequence groups shown below:

1. ESC d (simulates the Enter key)
2. Status Request (includes device control status)
3. Function Key/Device-Read format

Escape Sequence Programming

If you are using the first sequence (ESC d), you should consider using the approach shown in the example in the Correct Handshaking section. For the second and third sequences, you can use either of the two procedures described below, depending upon whether the length of the transmission is known to the program, or whether the length of the transmission may vary. Most status requests return a fixed number of characters, depending upon the request. The function keys and device-reads can return a text line of any length.

If the length of the transmission is known, the following Microsoft BASIC example works well. This example uses the INPUT\$ function to read the information. INPUT\$ does not echo the information, but must know how many characters to read (including any Carriage Return or Line Feed characters that may be returned):

```
1000 REM Get the current alpha cursor position
1010 PRINT CHR$(27);"a";CHR$(17);
1020 A$=INPUT$(12)
1030 PRINT MID$(A$,2)
```

You must accept 12 characters because the reported cursor position is sent in the form:

ESC & a nnn c mmm R <Carriage Return>

where 'nnn' is the column number and 'mmm' is the row number. Note that printing this same sequence would reposition the cursor to its current location. However, printing with the MID\$ function in line 1030 does not move the cursor.

If the length of the transmission can vary according to the item's current definition, you must use another approach. In the following BASIC example, the LINE INPUT statement echos characters if the application is reading from the standard input device. If the application is reading from a file, then it has no need to display the information on the screen. The following example assigns the CONSOLE (terminal) as an input file, and uses LINE INPUT# to read the information:

```
1000 REM Read the text in softkey #1
1010 PRINT CHR$(27);"&f1E";CHR$(17)
1020 OPEN "I",1,"CON"
1030 LINE INPUT #1,TEXT$
1040 CLOSE 1
```

The text in softkey 1 must end with a carriage return.

DATA OPERATIONS

This section describes escape sequences that control data transfer from a host system or an application program to the internal, external, and/or HPIB printers. A host system or an application program can send data already stored in display memory or data directly from the program to any selected printer. The following paragraphs describe the concepts and the escape sequences are listed after.

Selecting a Printer as the Destination Device

To control your printer, you must select it as the destination device for device control commands. The destination devices are selected programmatically by including one or more <d> parameters in the device control escape sequences (more than one destination device can be selected). The escape sequence to select destination devices is:

```
ESC &p <a> d <b> d <c> D
```

The values for <a>, , and <c> may be 4, 5, or 6. The option code "4" selects the printer indicated by the PrinterCode4 entry on the Terminal Configuration menu. The PrinterCode4 entry lets you choose between an external RS232 printer or the internal printer. The option code "5" selects an HPIB printer and the option code "6" selects the internal printer. If no <d> parameter is specified, then the currently assigned destination device(s) is used.

In addition to the above escape sequence, there is a level of system softkeys that lets you examine and select printer devices. You access the printer device softkeys by pressing the following key sequence:

- o [User System]
- o [device control] or [f1]
- o ["to" devices] or [f3]

This level of softkeys lets you select a serial device, the internal printer, or an HPIB device as the printer destination device. Using one of the escape sequence options (4, 5, or 6) alters the softkey selection accordingly.

NOTE

The "ESC & p <a> D" sequence usually puts an asterisk in the selected "to" device softkey label; however, an ESC & p 5D sequence will not cause an asterisk to be displayed on the "HPIB" softkey label.

Although the ESC & p 5D sequence does not display an asterisk in the "HPIB" softkey label, the HPIB printer is selected. If you press the HPIB "to" device softkey, you will get a "Device Busy" error message when you try to send data to the HPIB printer. To un-select the busy device, you can send an escape sequence to select a different device, such as the internal printer (ESC & p 6D).

Selecting a Source Device

To transfer data from the alphanumeric or graphics display memory to a selected printer, you must set the alphanumeric or graphics memory as the source device. Selecting alpha memory will cause data to be moved from alpha memory to the selected printer. Selecting graphics memory will result in a raster dump to the printer.

You can combine the source and destination assignments and the transfer initiation in one sequence:

ESC & p 7s 6d F

defines graphics memory as the source (7s), defines the internal printer (6d) as the destination, and copies the entire graphics display memory to the destination. The values for source devices and destination devices are listed later in this section.

Paper Movement

Two paper movement operations are allowed: line feed (advance line), and form feed (advance page). If the internal printer has been selected as the device, a form feed can be produced only in Report or Metric modes (see the next section on Printer Modes). If a form feed on an internal printer is attempted while not in Report or Metric mode, a line feed will be generated in place of the form feed.

Printer Modes

There are two printer modes: Record mode and Data Logging mode. They are available on the internal, external, and HPiB printers.

RECORD MODE: In Record mode, data is copied from the remote computer or a local application to the selected destination devices; data is not displayed on the screen. If Record mode is turned off with a partially-filled buffer, the contents of the buffer are sent to the destination device(s).

DATA LOGGING MODES: There are two Data Logging modes: Log Top mode, and Log Bottom mode. When alpha display memory is filled (48 lines), and one more line of data is received over a datacomm line, the top line in display memory is "purged" to make room for the new line.

With top logging, each line that is purged from the top of the display is sent to the selected printer. Thus, while the line is "lost" from display memory, it is kept in printed form.

With bottom logging, each time the cursor moves from one line to another line as the result of a line feed or an end-of-line wraparound, the line from which the cursor moved is printed. This feature allows you to maintain a hard copy "trail" of all lines added to display memory in the order they were received.

Escape Sequence Programming

Terminal to Printer Data Transfers

A selected line, all or part of a displayed page, or all or part of the active workspace (display memory) can be copied from the terminal to a printer. This can be done via the following escape sequence:

ESC &p <y>

The option values for the <y> parameter is listed later in the section on Data Transfer Escape Sequences. Printing begins at the current cursor line. You must select a printer as the destination device. When printing is initiated from the keyboard, the display is automatically defined as the source device.

COPY A SELECTED LINE: When either an external printer, the internal printer, or an HPIB printer is selected as a destination device, you can copy the line containing the cursor from the display to a printer.

COPY PAGE: When either an external printer, the internal printer, or an HPIB printer is selected as the destination device, you can copy all lines, starting with the line containing the cursor through the last line visible on the screen, to the printer.

COPY ALL: When either an external printer, the internal printer, or an HPIB printer is selected as a destination device, you can copy all lines, starting with the line containing the cursor through the last line of display memory, to the printer.

For the above operations, the internal printer is the only destination device that copies any escape sequence directly from the display to the printer. The serial printer or the HPIB printer will try to interpret the escape sequence instead.

In all copy operations, block terminators (carriage returns and line feeds) are ignored.

Computer to Printer Data Transfers

With the printer selected as a destination device, you can copy up to 256-character records from the computer to the printer. The data can be sent in either binary or ASCII form, depending on the escape sequence used.

Binary Data Transfer

If data is to be transmitted to a printer (or other external device) in binary (8-bit) form, the device must be of a type that accepts binary data. Also, the ENQ/ACK form of handshaking must be used. This means that the computer program which initiates the data transfer must transmit an ENQ (ASCII decimal 5); the terminal must respond with an ACK (ASCII decimal 6) before the data is transmitted.

The following escape sequence is used to initiate transfer of a string of binary data:

ESC &p <x>W

This sequence transfers the first "x" bytes of the data string, in binary form, to the printer. The maximum value for "x" is 256.

The sequence of events for transmitting five bytes of binary data ("12345") to an external device from a program, using the ENQ/ACK handshake, is shown below:

Computer program	Terminal
ESC&p5W	----->
ENQ	----->
	<----- ACK
"12345"	----->

As an alternative to the ENQ/ACK handshake, the program can pause for 3 seconds after sending the initiating escape code.

NOTE

If data is passed to the display with the eighth bit set, the system firmware will try to interpret it as one of its own internal display code. Therefore, if the display is selected as a destination device, sending binary data to it could cause unpredictable result.

Escape Sequence Programming

ASCII Data Transfer

To copy an ASCII data string from the computer to the printer, use the following escape sequence:

ESC &p W <data string>

The entire data string is copied. The string is terminated either by an ASCII line feed character or by the 256th character.

Data Transfer from HPIB Device

To transfer data from an HPIB device, use the following escape sequence to select the HPIB "talk" device address:

ESC &p <x>p 5u 1C

The parameter <x>p selects the HPIB address <x> as the "talk" device. (See the Data Transfer Escape Sequences section for additional information.)

Once you have selected the "talk" address, you can transfer data from the HPIB device (with the current "talk" address) to the computer by the following escape sequences:

For ASCII data terminated with a Line Feed (LF):

ESC &p 5s R

For ASCII data not terminated with a LF and binary data, send a byte count before transferring data. The terminal will not respond to any further program control until the specified number of bytes have been transferred.

Send Byte Count: ESC &p 5u <byte count>p 6C

Send Data: ESC &p 5s R

Determining Success of an Escape Sequence

After issuing a copy line, copy page, copy all, advance line, or advance page sequence, the remote program can determine whether or not the operation was successful by executing a BASIC INPUT statement, or similar instruction, that requests one ASCII character from the terminal. If the DC1 handshaking is used, you should send a DC1 character (decimal 17) to tell the terminal that you are ready to receive the operation status.

The terminal responds by sending "S", "F", or "U". "S" indicates successful completion, "F" indicates that the operation failed, and "U" indicates that the terminal operator interrupted the data transfer by pressing RETURN. "U" can be returned only for a copy page or copy all operation.

These completion codes cannot be suppressed by configuration parameters or any other means. They are always transmitted and your programs should include input commands for accepting them.

The keyboard is disabled ("locked") until the status is sent.

In either Character or Block Line mode, the terminal sends a <cr> (or <cr><lf> if Auto Line Feed mode is enabled) following the completion code. In Block Page mode, it sends a block terminator character.

If a data comm error occurs during transmission of a data record, the device control completion code is unpredictable. Datacomm errors are reported by way of the terminal status bytes described in the section on Status Requests.

Format Modes for the Internal Printer

The Internal printer permits you to specify three formats for printing: Compressed Characters, Report Format, and Metric Format.

COMPRESSED CHARACTERS MODE: The internal printer can print compressed characters in which each print line contains up to 132 characters spaced 16.2 to the inch. Once the printing of compressed characters is enabled, it remains enabled until it is disabled, until a hard reset is performed, or until the system power is turned off.

REPORT FORMAT MODE: The internal printer normally operates in continuous Format mode. You can, however, enable Report Format in which printing is output in 66-line pages with a 3-line top margin, 60 lines of text, and a 3-line bottom margin. The margins and text area together form an 8 1/2 by 11 inch page. The printer uses a small tic mark to mark the end of one page and the beginning of the next.

Once enabled, Report Format remains enabled until disabled, until a hard reset is performed, or until the system power is turned off.

METRIC FORMAT MODE: With Metric Format enabled, printing is output in 70-line pages with a 3-line top margin, 64 lines of text, and a 3-line bottom margin. The end of one page and the beginning of the next is indicated by a small tic mark. Once enabled, Metric Format remains enabled until disabled, until Report Format is enabled, until a hard reset is performed, or until the system power is turned off.

If a soft reset is performed while in Report Mode or Metric Format, the printer will skip three lines, print a tic mark, and skip three more lines.

Data Transfer Escape Sequences

The escape sequences to control data transfers on the HP 150 are:

ESC &p <x>S Selects device <x> as the source device.

<x> ---	DEVICE -----
3	Display screen
7	Graphics display (raster dump to selected destination device)

ESC &p <a>d d <c>D Selects device <a>, , and/or <c>
as the destination device.

<a>,,<c> -----	DESTINATION DEVICE -----
3	Display screen
4	Internal or external printer depending on the PrinterCode ⁴ entry on the Terminal Configuration Menu
5	An HPIB device such as a plotter or a printer with default address at 1 (This selection does not display an asterisk on the "to" device HPIB softkey label.)
6	Internal printer

Escape Sequence Programming

ESC &p <y>

Copies <y> amount of data to destination devices <a>, , and <c>. As many destinations as desired can be specified.

<y> ---	AMOUNT
b	The line in which the cursor is located (COPY LINE)
f	From the line in which the cursor is located to the last displayed line (COPY PAGE)
m	From the line in which the cursor is located to the end of display memory (COPY ALL)

ESC &p <x> ^

Requests the status of device <x>.
(Refer to the Section on Status Requests for more information.)

<x> ---	DEVICE -----
4	Internal or external printer depending on the PrinterCode4 entry on the Terminal Configuration menu
6	Internal printer

ESC &p <y>u <x>p <z>C

Performs the action specified on the selected destination device, <y>u.

<y>	DEVICE
---	-----

3	Display screen
---	----------------

4	Internal or external printer depending on the PrinterCode ⁴ entry on the Terminal Configuration menu.
---	------------------------------------------------------------------------------------------------------------------

6	Internal printer
---	------------------

<z>	ACTION
---	-----

0	Generates <x> form feeds
---	--------------------------

1	Spaces <x> lines
---	------------------

2-10	Generates <x> form feeds
------	--------------------------

11	Turns on Log Bottom mode
----	--------------------------

12	Turns on Log Top mode
----	-----------------------

13	Turns off any logging mode
----	----------------------------

14	Prints normal characters (Internal printer only)
----	--------------------------------------------------

16	Prints compressed characters (Internal printer only)
----	------------------------------------------------------

17	Turns on normal Report mode (Internal printer only)
----	-----------------------------------------------------

18	Turns on Metric Report mode (Internal printer only)
----	-----------------------------------------------------

19	Turns off any Report mode. (Internal printer only)
----	----------------------------------------------------

20	Turns on Record mode; <x> is the ASCII decimal value (1-127) used to end Record mode. Because <x> must be in the first character position on the line, it must be preceded by a Carriage Return and a Line Feed.
----	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Escape Sequence Programming

ESC &p <x>p 5u <z>C

Selects the HPIB "talk" and "listen" device address. The action is selected by <z>, as shown below:

<z> ---	ACTION -----
1	Selects <x> as the HPIB "talk" device.
2	Selects <x> as the HPIB "listen" device

ESC &p 5u <z>C

Initializes the HPIB device. The action is selected by <z>, as shown below:

<z> ---	ACTION -----
3	Enables HPIB timeout
4	Disables HPIB timeout
7	Initializes HPIB to power on configuration.

ESC &p 5s R

Transfers ASCII data that is terminated with a Line Feed from the HPIB device (with the current "talk" address) to the computer.

ESC &p 5u <x>p 6C

For ASCII data not terminated with a Line Feed and binary data, this sequence sends a byte count <x> before transferring data from the HPIB device (with the current "talk" address) to the computer.

ESC &p <x> W
<data string>

Transfers <x> bytes of the data string from the computer or a local application to the selected destination device in binary form (<x> is a decimal value in the range 1-256).

ESC &p W
<data string>

Transfers the data string, in ASCII form, from the computer or a local application to the printer selected as the destination device. The string is terminated either by the 256th byte or by an ASCII Line Feed character.

ESC &k <x>S

Enables Compressed, or Normal Character mode for the internal printer as designated by the character <x>.

<x>	ACTION
---	-----
0	Disables Compressed Character mode
2	Initiates Compressed Character mode

STATUS REQUESTS

A status request is a request made by an application to the terminal for information about the current condition of a physical device. A status request is sent in the form of an escape sequence; it has to be sent from a program. The terminal ignores any status requests that are sent directly from the keyboard. Once the terminal receives a status request, it returns a string of characters that contains the requested information. This information is in the form of a data block, which is similar to data blocks that are generated in Block mode. The following BASIC program segment shows the operation of a terminal secondary status request:

```
10 PRINT CHR$(27)+"~";CHR$(17);
20 LINE INPUT A$
```

Statement 10 sends the ESC ~ sequence to request secondary status and sends the DC1 character to inform that it is ready to accept data. Statement 20 stores the information returned by the terminal in the string variable, A\$.

There are seven types of status requests:

1. **TERMINAL IDENTIFICATION.** This request is the means by which your program determines the kind of terminal it is communicating with.
2. **TERMINAL FEATURES.** There are four types of terminal-features requests: alphanumeric capabilities, graphics capabilities, interface capabilities, and the amount of Random Access Memory (RAM).
3. **PRIMARY TERMINAL STATUS.** This request returns seven bytes that report the status of some of the mode keys (such as Caps, Remote, Block, etc.), various error and pending flags, and the following Terminal Configuration menu fields:

```
XmitFncn(A)
SPOW(B)
InhEolWrp(C)
Line/Page(D)
InhHndshk(G)
InhDC2(H)
```

4. **SECONDARY TERMINAL STATUS.** This request returns seven bytes that report the status of the memory lock, buffer memory, and I/O firmware.
5. **DEVICE STATUS.** This request returns three bytes that report the status of the internal printer, HPiB, and/or RS232 device.
6. **CURSOR-POSITION SENSING.** This request returns an escape sequence that contains the row and column in which the cursor is located. Cursor position sensing is described earlier in this section.
7. **COMMAND COMPLETION-STATUS.** After issuing a copy line, copy page, copy all, advance line, or advance page sequence, the terminal automatically returns one character that indicates the completion status of the operation. The returned character indicates satisfactory completion (S), failure (F), or interruption of the operation (U) because the terminal operator pressed the [Return] key.

The following paragraphs describe the escape sequence used for each of the above requests, and also the format of the returned status information.

Status-Transfer Handshaking

The terminal treats all status requests as block transfers. In response to a status request, the terminal transmits the following items: an escape sequence, a series of data bytes, and a terminator. The terminator varies according to the mode as shown below:

Character Mode: <CR> or <CR><LF>

Block Line Mode: <CR> or <CR><LF>

Block Page Mode: <Block Terminator>

In either Character mode or Block Line mode, the terminator is <CR><LF> if Auto Line Feed mode is enabled. In Block Page mode, the block terminator is as selected on the Terminal Configuration menu. The default block terminator is <RS>.

The type of handshaking used is determined by the G and H straps described in the section on Block Transfers. The user can also set the G and H straps from the keyboard, by setting the InhHndShk and Inh DC2 fields of the Terminal Configuration menu as shown below:

Table 2-7. Setting the Handshaking Protocol

InhHndshk(G) -----	Inh DC2(H) -----	Handshake -----
No	Yes or No	DC1
Yes	Yes	No Handshake
Yes	No	DC1/DC2/DC1

NOTE

A status-request escape sequence resets the "block trigger received" flag. For example, if you are using the DC1 handshake and the terminal receives a <DC1> followed by the request, the terminal "forgets" that a block trigger was just received; therefore, the terminal will not send the data immediately. The terminal will not start the data transfer unless it receives another <DC1>.

Status-Transfer Priority

When handshaking is in effect and the terminal receives more than one status request, status-data transfers are constructed and sent according to the block-transfer-priorities shown below. Only one status transfer occurs for each complete handshake, although more than one may be pending.

Priority of Block Transfers

```

highest Primary status (ESC ^ )
.      Secondary status (ESC ~)
.      Device status (ESC &p <n> ^ )
.      Cursor sense (ESC ` or ESC a)
.      Transmit user-defined softkeys (f1-f8)
.      Display transfer ([ENTER] key or ESC d)
.      Command completion status (S, F, or U returned)
lowest  Terminal ID and capabilities (ESC *s ^ )
  
```

If the terminal receives more than one status request of the same type, once the handshaking is completed, the terminal acknowledges and responds only to the most recently received request.

Terminal Identification

You request the terminal ID status by sending the following escape sequence:

```
ESC * s ^
```

The terminal responds by sending back the following string:

```

2623A
or
150A
  
```

depending on the setting for "Terminal ID" in the Terminal Configuration menu. If you are communicating with an HP 3000, you should specify 2623A. You can change and save this value manually by going to the Terminal Configuration Menu.

Interpreting the Status

The terminal responds to all status requests by returning an escape sequence followed by a string of bytes. The lower bits of each byte contains the status information. The upper four bits are set so that the byte translates into an ASCII printing character (that is, a character that has an ASCII decimal value from 32 through 126). For example, the format for primary, secondary, and device status requests is shown in the table below. Notice that the upper four bits of each byte are set to "0011", which limits the ASCII decimal values to a minimum value of 48 (for these requests), well within the range of ASCII printing characters. The format for terminal capabilities status bytes is similiar.

Table 2-8. Returned Status Byte Format

ASCII Character	Binary Value
0	0011 0000
1	0011 0001
2	0011 0010
3	0011 0011
4	0011 0100
5	0011 0101
6	0011 0110
7	0011 0111
8	0011 1000
9	0011 1001
:	0011 1010
;	0011 1011
<	0011 1100
=	0011 1101
>	0011 1110
?	0011 1111

Terminal Status

The terminal status is indicated by 14 status bytes (0-13) that contain information about the display memory size, the strap settings for the configuration menu, and any terminal errors. These 14 status bytes are displayed, in two groups of seven bytes each, below the system-test screen pattern when [SYSTEM TEST] softkey is pressed. The first seven status bytes (0-6) correspond to the primary terminal status, and the second seven bytes correspond to the secondary terminal status.

PRIMARY TERMINAL STATUS.

You request the first set of terminal status bytes (bytes 0-6) by sending the following escape sequence:

ESC ^

The terminal responds with an ESC \, and seven status bytes followed by a terminator to the application. Figure 2-1 shows a typical primary terminal-status request and response. This example assumes that the DC1 handshake is being used and that the appropriate terminator is a <CR>. Figure 2-2 illustrates the function of each bit in each byte.

Escape Sequence Programming

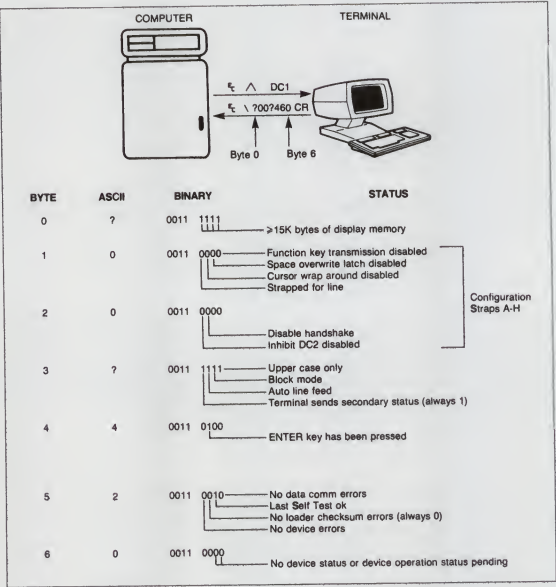


Figure 2-1. Terminal Primary Status Example

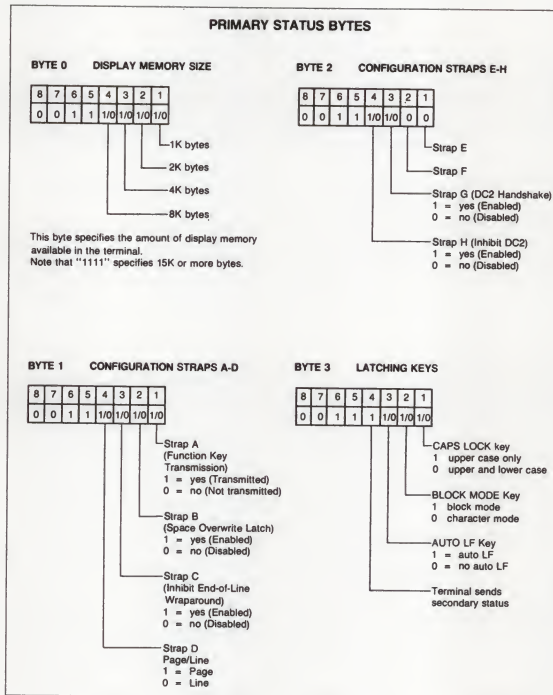


Figure 2-2. Terminal Primary Status Bytes

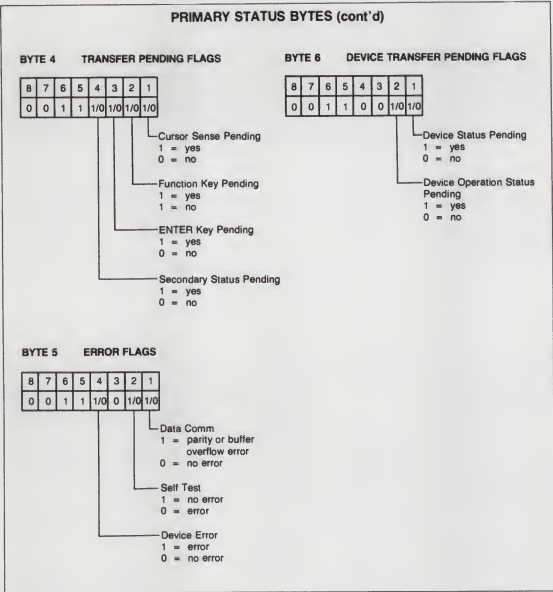


Figure 2-2. Terminal Primatry Status Bytes (Continued)

SECONDARY TERMINAL STATUS.

You request the second set of terminal-status bytes (bytes 7-13) by sending the following escape sequence:

ESC ~

The terminal responds with an ESC |, and seven status bytes followed by a terminator. Figure 2-3 shows a typical secondary terminal status request and response. This example assumes that the DC1 handshake is being used and that the appropriate terminator is a <CR>. Figure 2-4 illustrates the function of each bit in each byte.

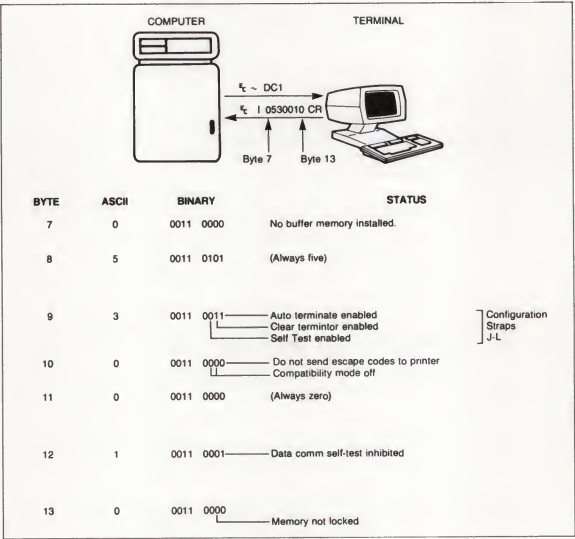


Figure 2-3. Terminal Secondary Status Example

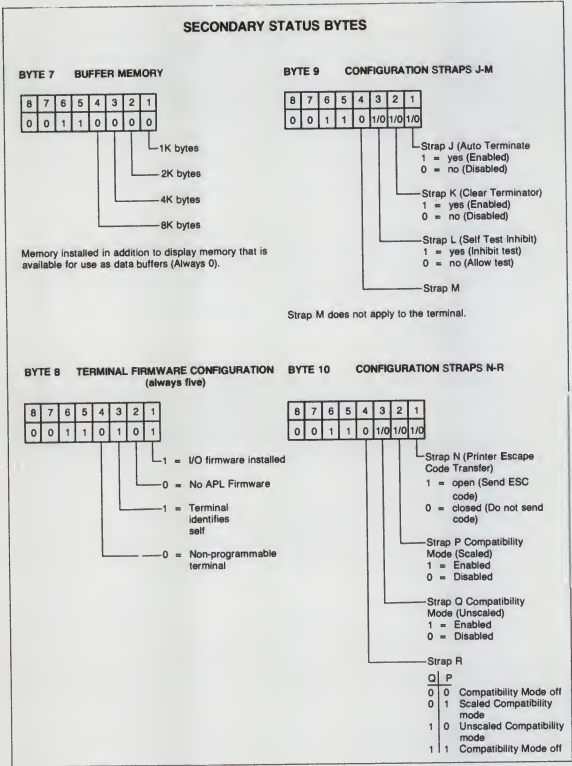


Figure 2-4. Terminal Secondary Status Bytes

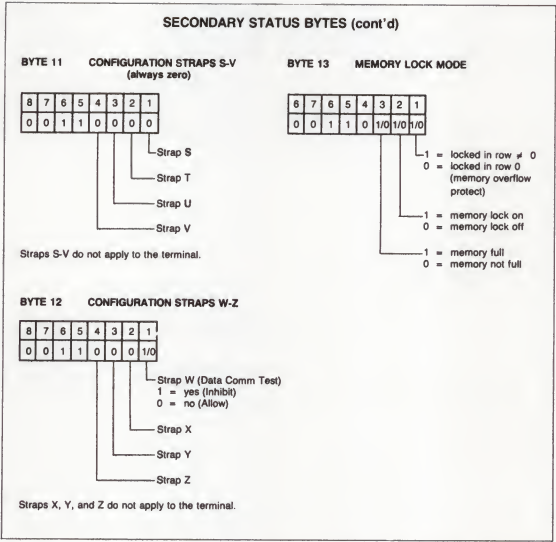


Figure 2-4. Terminal Secondary Status Bytes (Continued)

TERMINAL CAPABILITIES.

Four requests can be issued for terminal capabilities: alphanumeric, graphics, interface capabilities, and the amount of terminal RAM. These requests are generated with the following escape sequence:

```
ESC *s <x> ^
```

where "x" selects the request type, as follows:

Table 2-9. Terminal Capability request

X	REQUESTED INFORMATION
---	-----
-1	Alphanumeric capabilities
-2	Graphics capabilities
-3	Amount of Terminal RAM memory
-4	Interface capabilities

Escape Sequence Programming

The terminal responds with a string of bytes. The first byte indicates the number of status bytes in the response. (This byte is not included in the count.) The following byte or bytes contain the requested data as shown in (figures 2-5 through 2-9).

If the "x" parameter is less than -4 (-5, -6, etc.), then a single byte indicating 0 status bytes is returned. (This byte is the "@" character (01000000).) If "x" is greater than 1 and if the terminal contains graphics, then the terminal returns graphics-status information.

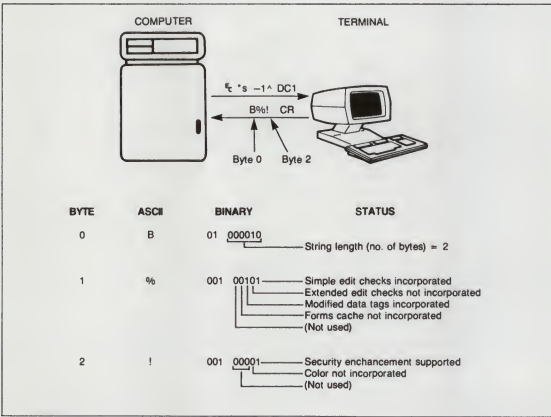


Figure 2-5. Terminal Capabilities (Alphanumeric-Typical) Status Example

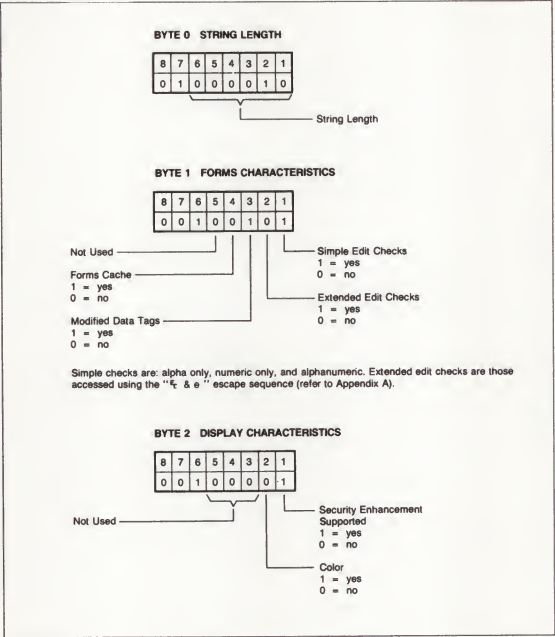


Figure 2-6. Terminal Alphanumeric Capabilities Status Bytes

Escape Sequence Programming

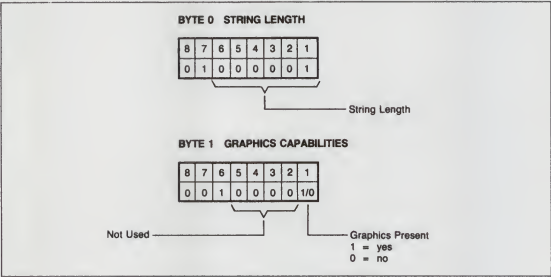


Figure 2-7. Terminal Graphics Capabilities Status Bytes

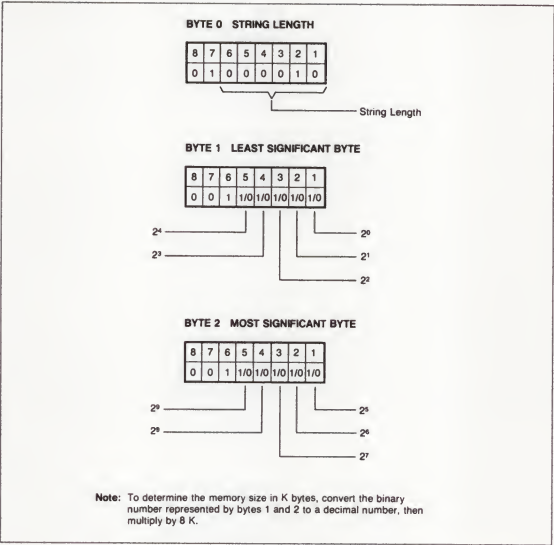


Figure 2-8. Installed Memory Status Bytes

Escape Sequence Programming

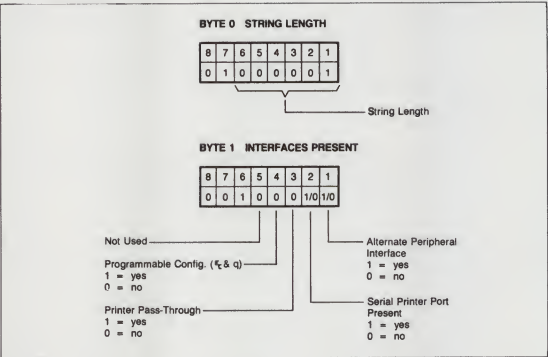


Figure 2-9. Terminal Interface Capabilities Status Bytes

Device Status

You can obtain information about the status of the internal printer or external device by issuing a device-status request. This request would typically be made following a print operation or after examining bytes 5 and 6 of the terminal status.

You request the device-status by sending the following escape sequence:

`ESC &p <device code> ^`

where <device code> is either 4 or 6. If <device code> is 6, the terminal returns status on the internal printer. If <device code> is 4, then the terminal returns information about the external device or internal printer depending on the setting of the PrinterCode4 field in the Terminal Configuration menu. The PrintCode4 field lets you choose between an external device or the internal printer.

If <device code> is any value other than 4 or 6, the escape sequence is ignored.

The terminal responds with the sequence `ESC \ p<device code>`, followed by three status bytes followed by a terminator. Figure 2-10 shows a typical device status request and response.

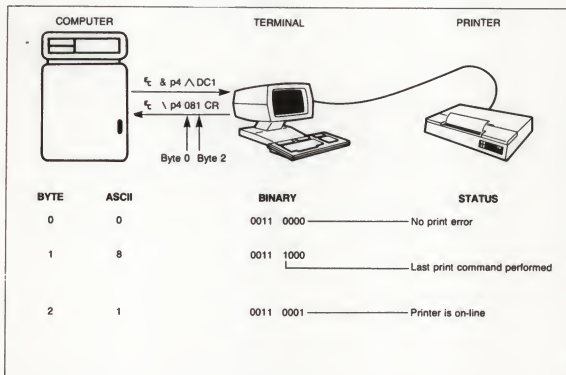


Figure 2-10. Device Status Example

Escape Sequence Programming

The device status bytes are shown in figure 2-11.

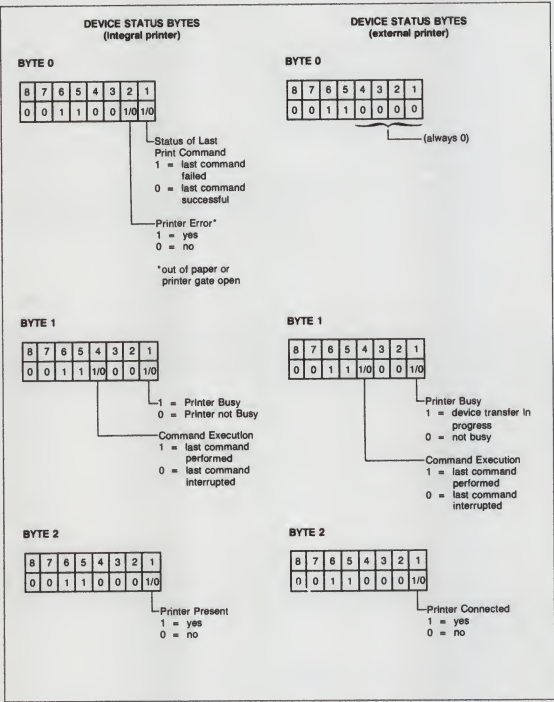


Figure 2-11. Device Status Bytes

ALPHA/GRAPHICS I/O SUBSYSTEM

SECTION

3

NELSYS THT - EL

INTRODUCTION

The Alphanumeric and Graphics Input/Output Subsystem (AGIOS) is a set of system functions that lets you access the alphanumeric and graphics intrinsics of the HP 150. These function calls let you perform text and graphics mode operations on your display, let you control your keyboard, let you define and use application softkeys, and let you perform all touchscreen operations.

AGIOS provides an efficient and rapid way of accessing terminal features that is five times faster than using comparable escape sequences. In addition, AGIOS provides functions such as transferring a block of data and enhancements to the display, defining application softkeys, intercepting keystrokes, and defining characters in Graphics Text mode, which can not be done with escape sequences.

AGIOS can best be programmed with Intel 8086/8088 assembly language. If you must access AGIOS functions from a high-level language, such as BASIC or Pascal, you may have to link an assembly module which has the desired AGIOS functions and a high-level language module which calls the assembly functions. (See Appendix B on AGIOS Function Calls from High-Level Languages for more information.)

You can access each AGIOS function by performing the following steps:

1. Select the desired function
2. Supply the required input parameters
3. Invoke AGIOS by issuing a software interrupt
4. Check the completion code returned by AGIOS

You might at first think that AGIOS functions are cumbersome to use. But you can write a general routine to set up all the common parameters that AGIOS requires to execute a function. Whenever you are ready to make a call to AGIOS, you set up those parameters that are unique to the particular function. Then you can call the general routine to handle the rest of the routine work of calling AGIOS for you.

When using AGIOS functions, there is one thing you should remember. The I/O subsystem does not perform any error checking on input parameters. These functions assume that all input parameters given are valid. Therefore, before calling AGIOS, you should check that all input parameters to the I/O subsystem are legal. (There are examples throughout the AGIOS sections that show you how to use AGIOS functions.)

Alpha/Graphics I/O Subsystem

The alphanumeric functions are listed in the order of function number in the Alphanumeric Input/Output Subsystem (AIOS) section. The graphics functions are listed in the order of function number in the Graphics Input/Output Subsystem (GIOS) section. For each AGIOS function, there are detailed descriptions of input parameters, output status, and additional remarks.

Where applicable, the AGIOS call name is followed by the corresponding escape sequence. For example,

```
DEFINE TOUCH FIELD (ESC - z g)
```

This indicates that the escape sequence which corresponds to the Define Touch Field AGIOS call is ESC - z g. Additional parameters may be required. (See the section on Escape Sequence Programming for the exact syntax.)

In the video intrinsics subsection, a sample assembly program segment is included with a few selected functions to illustrate the use of these functions. Once you understand those sample program segments, calling any AGIOS functions should be straight forward. Some additional Pascal callable video intrinsics examples are included in Appendix C. You will also find a quick reference on AGIOS functions and equivalent escape sequences in Appendix A.

HOW TO USE AGIOS FUNCTION CALLS

You access AGIOS calls via MS-DOS system function 44 hex. An AGIOS call is an "I/O Control Write" on the console device. (See the *Microsoft Programmer's Reference Manual* on system function 44 hex).

In general, the parameters of each AGIOS function are passed in a buffer in the order specified in each function's description.

For each function, the first parameter is a 2-byte function code. The high byte is the index number (0 = AIOS, 4 = GIOS); the low byte is the function number. The function code is given in decimal.

If additional parameters are required, they will be specified after the function code. Your application has to build a buffer with the required parameters stored in the order that they are listed in each function's description. The address (segment and offset) of this input buffer is then passed to the AGIOS function.

Each AGIOS function is initiated by setting up proper Intel 8088 registers and then issuing an interrupt to the operating system (INT 21H).

The 8088 registers should be set up as follows:

- AX = I/O Control Write (4403H)
- BX = Console device handle (1)
- CX = Number of bytes in buffer
- DS = Buffer segment address
- DX = Buffer offset address

Upon return from an AGIOS function, the AX register contains the completion status. A zero value in AX indicates the function was successful; otherwise, the function was not successful. Your application should always examine the AX register after each AGIOS call for the completion status.

NOTE

AIOS function call (0,25) in Operating System version B.01.00 and later returns an address that you can call to execute AGIOS functions instead of using system function 44 hex. Calling this address eliminates the MS-DOS system function overhead; therefore, AGIOS functions are executed faster. (See function (0,25) in the section on Alphanumeric Subsystem Function Reference for detailed information.)

AN EXAMPLE ON CALLING AN AGIOS FUNCTION

The following example shows how you use AGIOS function calls in assembly language. The function call is the Execute Two-Character Sequence, which is used here to clear the display screen. This AGIOS function call requires a three-byte buffer that contains the function code followed by the "J" parameter:

BUFF	16 0 J
Byte:	+0 +1 +2

A sample assembly program segment calling the AIOS function to clear the display is listed below:

```

CLRSCRN:  MOV  AX,4403H      ;I/O control write
           MOV  EX,1         ;console handle, always = 1
           MOV  CX,3         ;BUFF Length = 3
           MOV  DX,OFFSET BUFF ;BUFF offset address
                               ;segment address is already in DS
           INT  21H          ;execute the function
           RET               ;return
BUFF      DB   16,0,'J'     ;BUFF content

```

In the above example, BUFF data segment is assumed to have already been assigned to DS.

A 'clear display' operation can be accomplished with a simple 'PRINT CHR\$(27) + "J"' statement in BASIC or a similar print statement in other languages. However, the AGIOS call executes the 'clear display' function faster. Moreover, the AGIOS function you want to execute may not have an equivalent escape sequence associated with it.

As a result, you will want to use AGIOS functions in applications that require a faster and more efficient method of programmatically controlling the HP 150 system.

SYNTAX USED IN AGIOS FUNCTION CALLS

Each AGIOS function call is explained in detail in the sections on AGIOS function reference. Each function requires a number input parameters stored in a buffer to be passed to the function. In order to clarify the parameters your application needs to supply, a standard notation is used in the function reference sections. Before you try to use these AGIOS functions, you should understand the syntax used.

One simple method to set up parameters and pass them in a buffer is to define them using Microsoft Macro Assembler directives DB (Define Byte), DW (Define Word), or DD (Define Double Word). These directives are discussed in the *Microsoft Macro Assembler Manual*.

Parentheses, positional notation, and possible implementation of assembler directives are used as follows:

PARAM	Indicates a single byte (8 bit) parameter.	DB PARAM
(PARAM1,PARAM2)	Indicates two single byte parameters with param1 in the high byte and param2 in the low byte.	DB PARAM2,PARAM1
(,PARAM)	Indicates a single byte parameter in the low byte of the word. The high byte is ignored.	DB PARAM,0
(PARAM,)	Indicates a single byte parameter in the high byte of the word. The low byte is ignored.	DB 0,PARAM
(PARAM)	Indicates a word (16 bit) parameter.	DW PARAM
((PARAM))	Indicates a double word (32 bit) parameter. Usually the low word is a data offset address and the high word is a segment address.	DD PARAM

If ((PARAM)) is used as a 32-bit address, you could define it using the Define Word (DW) assembler directive:

```
DW PARAM_OFFSET,PARAM_SEGMENT
```

In the above instruction, PARAM_OFFSET is the 16-bit offset address and PARAM_SEGMENT is the 16-bit segment address.

NOTE

If a variable is defined as a word, the two bytes are stored in memory backward; that is, the least significant byte (or low byte) is stored in memory before the most significant byte (or high byte).

ALPHANUMERIC INPUT/OUTPUT SUBSYSTEM FUNCTION REFERENCE

SECTION

4

NELSI THT - EL

INTRODUCTION

This section lists functions in the Alphanumeric Input and Output Subsystem (AIOS) in numerical order according to the function number. If you do not already know the procedure for calling Alphanumeric/Graphics Input/Output Subsystem (AGIOS) functions and are not familiar with parentheses and positional notations used in the function reference, you should read Section 3 on AGIOS first.

AIOS FUNCTIONS

Each AIOS function is identified by a 16-bit function code as shown below:

Bit	15	-----		0
		0	X	

	high		low	
	byte		byte	

For AIOS functions, the high byte of the function code is always 0. The low byte (X) is the function number. The function code must be the first parameter stored in the input buffer.

In this section, an AIOS function code is given in decimal and is represented by an ordered-pair (0,X). You should note that the Intel processor interprets the two bytes in a word in reverse order. That is, in memory, the value X is stored before the value 0.

There are two ways of defining a function code. You can define a function code either in byte-mode or in word-mode using Microsoft Macro Assembler directives Define Byte (DB) or Define Word (DW).

For example, to allocate a buffer called IN_BUF and store the function code (0,1) in the first word of IN_BUF, you can use either one of the following instructions:

In Byte-mode: IN_BUF DB 1,0

In Word-mode: IN_BUF DW 1H

AIOS Function Reference

AIOS lets you programmatically control the following alphanumeric functions:

- o Video
- o Application Softkeys
- o Keyboard and Display Control
- o Touchscreen
- o Keyboard Processing

BATCH FUNCTION CALL

A special function call is available which lets you execute a sequence of function calls automatically. Using this function call is especially convenient when you frequently perform the same set of AGIOS function calls.

To "batch" function calls, you set up the sequence of AGIOS function calls in a *command buffer*. Then you issue the following batch function call using the command buffer as one of its parameters.

Input Buffer:

(0,0)	Function code
(BUFFER-LENGTH)	The number of bytes in the command buffer
((COMMAND-BUFFER))	Segment and offset address of the buffer containing the AGIOS function calls. The function calls are defined consecutively. Each function's parameter format is the same as specified for the individual function call.
	First word = Command buffer offset address
	Second word = Command buffer segment address

Output Status:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

A batch call is aborted when any of the function calls in the command buffer causes an error condition, but any functions specified before the error function will be executed. Additionally, you cannot nest batch function calls; that is, you cannot include this function in the command buffer.

Example:

This example clears the alpha display by homeing up first, then clearing the display. Refer to the 'H' and 'J' options of the "Execute Two-Character Sequence" function call (0,16) in this section.

The command buffer looks like this:

Contents -->	16 0 H 16 0 J
Byte -->	+0 +1 +2 +3 +4 +5

AIOS Function Reference

The assembly routine that sets up the command buffer and executes a batch call might look like this:

```
CLS:  MOV  AX,CS                ; assign data segment to be
      MOV  DS,AX                ; the same as the code segment
      MOV  CMDSEG,AX            ; and store it in input buffer
;
      MOV  AX,4403H             ; I/O control write
      MOV  BX,1                 ; console handle
      MOV  CX,8                 ; input buffer length
      MOV  DX,OFFSET BATCH      ; input buffer offset address
      INT  21H
      RET
;
CMDBUF DB 16,0,'H',16,0,'J'    ; command buffer content
BATCH  DB 0,0                  ; batch command function code
BUFLen DW 6                    ; command buffer length
CMDOFF DW CMDBUF               ; command buffer offset
CMDSEG DW ?                    ; command buffer segment
; (value to be assigned)
```

VIDEO INTRINSICS

The video intrinsics are a set of functions that can be used to manipulate the information (characters, enhancements, and character sets) displayed on the screen.

This section gives the AIOS function calls that support the video portion of the HP 150 system. The following video functions are explained in detail in this section.

<u>Function Code</u>	<u>Function</u>
(0,1)	Define Area
(0,2)	Write Area
(0,3)	Clear Area
(0,4)	Enhance Area
(0,5)	Read Area
(0,6)	Shift Area
(0,7)	Write Line

With the exception of the Write Line function, all video intrinsics operate on a defined area of the display. The total size of the display is 48 rows by 80 columns. However, only 24 of the rows are visible on the screen at any given time. The rows are numbered from 0 through 47; the columns are numbered from 0 through 79.

A defined area can be from 1 to 48 rows high, and from 1 to 80 columns wide. That is, the defined area can be as small as one character (one row high and one column wide) or as large as the entire display (48 rows high and 80 columns wide).

The speed of the video intrinsics varies directly with the size of the defined area. The larger the area, the faster the video intrinsics operate. You will obtain the most efficient performance from the video intrinsics if your defined area is as large as is practical for your application.

There is a one-to-one correspondence between the position of a data byte in its buffer and the character position that it will affect in the defined area, starting at the upper left corner of the defined area, incrementing column position first and then row position.

The character data consists of the 8 bit extended ASCII character code (Roman-8).

AIOS Function Reference

The HP 150 has four different display enhancements. Characters can appear in the following ways, or in any combination of them:

- o Half-bright
- o Underline
- o Inverse
- o Blinking

The following table shows the different combinations of display enhancements and enhancement codes. The enhancements range from '@' (40H) to '-' (5FH). A space (20H) indicates that the currently active enhancement is used.

Display Enhancements Table

Security off:	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Security on :	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	-	
Half-Bright									X	X	X	X	X	X	X	X
Underline					X	X	X	X					X	X	X	X
Inverse Video			X	X			X	X			X	X			X	X
Blinking		X		X		X		X		X		X		X		X
No Enhancement	X															

If a "security on" enhancement is used, the enhancement is displayed, but the character is hidden.

The character set and the code associated with each set is listed below:

Code	Character Set
@	Normal Roman
A	Line Drawing
B	Bold Face Roman
C	Italic Roman
D	Math
SPACE	The current character set

As described in the Escape Sequences section, the "ESC) <cset>" sequence can be used to select alternate character set. However, this escape sequence only allows you to select the Normal Roman, Math, and Line sets. In order to use Bold Face and Italic sets, you have to use video intrinsics in AGIOS.

NOTE

A null data buffer pointer (segment = OFFFFH) will suppress the update operation for that data type.

Define Area

This function specifies the area to be operated upon by subsequent video intrinsics.

Input Buffer:

(0,1)	Function Code.
(LR-ROW,LR-COL)	Defines the lower right corner of the defined area. LR-ROW = Lower right row (0-47) LR-COL = Lower right column (0-79)
(UL-ROW,UL-COL)	Defines the upper left corner of the defined area. UL-ROW = Upper left row (0-47) UL-COL = Upper left column (0-79)
((PREV-COORD))	A double-word pointer to a buffer where the previous defined area coordinates are returned. First word = Buffer offset address Second word = Buffer segment address If this pointer is null (segment = OFFFFH), then these coordinates will not be returned.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful
PREV-COORD	The coordinates are returned in four consecutive bytes: lower right column, lower right row, upper left column, upper left row.

Remarks:

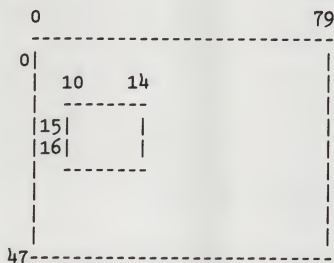
You should call this function before calling any of the other video intrinsics functions, with the exception of the Write Line function. You can use this function at any time to change the defined area. Once you have defined an area, you need not call this function again unless you want to change the size or the location of the defined area.

Note that the speed of the video intrinsics varies directly with the size of the defined area. The larger the defined area, the faster the video intrinsics operate.

AIOS Function Reference

Example:

This example defines a rectangular area from column 10 to column 14 and from row 15 to row 16.



Function Code = (0,1)
(LR-ROW,LR-COL) = (16,14)
(UL-ROW,UL-COL) = (15,10)
((PREV-COORD)) = ((-1))

The following is a sample assembly program segment which calls the Define Area function to define such an area:

```
; Build the input buffer with the specified parameters.
IN_BUF  DW  1                ;function code
        DB  14               ;lower right column
        DB  16               ;lower right row
        DB  10               ;upper left column
        DB  15               ;upper left row
        DW  OFFFFH           ;don't return
        DW  OFFFFH           ;previous coordinates
        .
        :
```

; Set up registers for AGIOS call.

; DS register is assumed to be the buffer's data segment.

```
;
EUNICE: MOV  AX,4403H         ;I/O control write
        MOV  BX,1            ;console handle
        MOV  CX,10           ;buffer length
        MOV  DX,OFFSET IN_BUF ;buffer offset address
        INT  21H             ;call Define Area
        RET                  ;return
```

See a Pascal-callable example of this function in Appendix C.

Write Area

Three items of display information are associated with each character cell in the display. These items are the character itself, the enhancement (if any), and the character set.

The Write Area function uses the data in three buffers to update the display information. There is a one-to-one correspondence between the position of the data byte in each buffer and the affected character-cell position in the defined area on the screen. The first byte in each buffer corresponds to the upper left-hand corner of the defined area. The information in this buffer increments the column positions first, and then increments the row positions.

Input Buffer:

(0,2)	Function Code.
(DATA LENGTH)	Length of the data buffers.
((ENH POINTER))	A double-word pointer to a buffer of enhancement codes to map to character cells in the defined area.
	First word = Buffer offset address
	Second word = Buffer segment address

Display Enhancements Table

Security off:		@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Security on :		P	Q	R	S	T	U	V	W	X	Y	Z	[\]		-

Half-Bright											X	X	X	X	X	X	X
Underline						X	X	X	X					X	X	X	X
Inverse Video				X	X			X	X			X	X			X	X
Blinking			X		X		X		X		X		X		X		X
No Enhancement		X															

If this pointer is null (segment=OFFFH), the enhancements are not changed.

AIOS Function Reference

((CHAR SET POINTER)) A double-word pointer to a buffer of character set codes to map to character cells in the defined area.

First word = Buffer offset address
Second word = Buffer segment address

<u>Code</u>	<u>Character Set</u>
@	Normal Roman
A	Line Drawing
B	Bold Face Roman
C	Italic Roman
D	Math
SPACE	The current character set

If this pointer is null (segment=OFFFFH), then the character sets are not changed.

((CHAR POINTER)) A double-word pointer to the buffer of character data to map to character cells in the defined area.

First Word = Buffer offset address
Second Word = Buffer segment address

If this pointer is null (segment=OFFFFH), then the characters are not changed.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Before you use this function, you must use the Define Area function (0,1) to define the display area first.

If you want to specify one enhancement for the entire defined area, you should use the Enhance Area function (0,4). Otherwise, the system's performance will be slowed because of the time that the system needs to process the enhancement buffer.

If all Write Area functions in your application use no enhancement and Normal Roman character set, you can use the Clear Area function to initialize the defined area. You can then set ENH POINTER and CHAR SET POINTER to null (segment = OFFFFH) for all the Write Area functions in your application.

***** See an example of this function on the next page *****

Example:

This example assumes that the defined area has already been defined. (See the example in the Define Area function (0,1).) This example calls the Write Area function to write 'HELLO THERE' in inverse blinking in the defined area.

	10	11	12	13	14	

15		H	E	L	L	O
16		T	H	E	R	E

Function Code = (0,2)
 (DATA LENGTH) = (10)
 ((ENH POINTER)) = 32-bit pointer
 ((CHAR SET POINTER)) = ((-1))
 ((CHAR POINTER)) = 32-bit pointer

The following is a sample assembly program segment which sets up the above input parameters and calls Write Area to write the data into the defined area. The enhancement and character buffers' segment addresses are assumed to be the same as the current data segment address.

```

; Build the input buffer
IN_BUF    DW 2                ;function code
          DW 10               ;data buffers length
          DW ENH_PTR          ;enhancement offset
SEG1       DW ?               ;segment to be assigned later
          DW -1               ;character set
          DW -1               ;does not change
          DW CHAR_PTR         ;character offset address
SEG2       DW ?               ;segment to be assigned later
ENH_PTR    DB 'CCCCC'         ;enhancement buffer
          DB 'CCCCC'         ;inverse blinking = C
CHAR_PTR   DB 'HELLO'        ;character
          DB 'THERE'         ; buffer
          .
          .
          .
; Set up registers for AGIOS call.
; DS register is assumed to be the buffer's data segment.
MOV AX,DS      ;get current data segment address
MOV SEG1,AX    ;assume buffer segment address
MOV SEG2,AX    ; is the same as data segment address
MOV AX,4403H   ;I/O Control Write
MOV BX,1       ;console handle
MOV CX,16      ;input buffer length
MOV DX,OFFSET IN_BUF ;buffer offset address
INT 21H        ;call Write Area
RET            ;return

```

See a Pascal-callable example of this function in Appendix C.

Clear Area

This function clears a defined display area to ASCII blanks (20H), no enhancements, and Normal Roman character set.

Input Buffer:

(0,3)	Function Code
-------	---------------

Outputs:

AX = 0	Successful
AX = <> 0	Unsuccessful

Remarks:

Before you use this function, you must use the Define Area function (0,1) to define a display area first.

See a Pascal-callable example of this function in Appendix C.

Enhance Area

This function sets the specified enhancement to the entire defined display area.

Input Buffer:

(0,4) Function Code.

(,ENHANCEMENT) An enhancement code from '@' (40H) through
 '_' (5FH) (The high byte is a "don't care".)

Display Enhancements Table

Security off:	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Security on :	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	-	
Half-Bright									X	X	X	X	X	X	X	X
Underline					X	X	X	X					X	X	X	X
Inverse Video			X	X			X	X			X	X			X	X
Blinking		X		X		X		X		X		X		X		X
No Enhancement	X															

Outputs:

AX = 0 Successful

AX <> 0 Unsuccessful

Remarks:

Before you use this function, you must use the Defined Area function (0,1) to define a display area first.

This function does not affect characters and character sets already in the defined area. If you choose to use the "security" enhancements, then the characters in the defined area will not be displayed until you turn off security.

See a Pascal-callable example of this function in Appendix C.

Read Area

Three items of display information are associated with each character cell in the display. These items are the character itself, the enhancement (if any), and the character set. The Read Area function reads the display information, and returns it in three buffers. This function fills each buffer with the applicable information, which it reads from right to left across the defined area (one row at a time, starting with top row in the defined area).

Input Buffer:

(0,5) Function Code

(DATA LENGTH) Length of the data buffers.

((ENH POINTER)) A double-word pointer to the buffer that the enhancements data will be returned.

First word = Buffer offset address
Second word = Buffer segment address

Display Enhancements Table

Security off:	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Security on :	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	-	
Half-Bright										X	X	X	X	X	X	X
Underline					X	X	X	X					X	X	X	X
Inverse Video			X	X			X	X			X	X			X	X
Blinking		X		X		X		X		X		X		X		X
No Enhancement	X															

If this pointer is null (segment=OFFFFH), then the enhancements are not returned.

((CHAR SET POINTER)) A double-word pointer to a buffer that the character sets will be returned.

First word = Buffer offset address
Second word = Buffer segment address

<u>Code</u>	<u>Character Set</u>
@	Normal Roman
A	Line Drawing
B	Bold Face Roman
C	Italic Roman
D	Math

If this pointer is null (segment=OFFFFH), then the character sets are not returned.

((ASCII POINTER)) A double-word pointer to the buffer that the character data will be returned.

First word = Buffer offset address
Second word = Buffer segment address

If this pointer is null (segment=OFFFHH), then the character data is not returned.

Outputs:

AX = 0 Successful
AX <> 0 Unsuccessful

ENH The buffer of enhancements in the defined area
CHAR SET The buffer of character set codes in the defined area
ASCII The buffer of character data in the defined area

Remarks:

Before you use this function, you must use the Defined Area function (0,1) to define a display area first.

This function will return the number of bytes specified in DATA LENGTH regardless of the actual size of data buffers. However, if DATA LENGTH is greater than the number of character cells in the defined area, then only the data in the defined area is read.

See a Pascal-callable example of this function in Appendix C.

Shift Area

This function shifts the characters, enhancements, and character sets in the defined area by a specified number of rows or columns. This function also lets you assign display information for the area vacated by the action of shifting the data in the defined area.

Input Buffer:

- (0,6) Function Code
- (DATA LENGTH) Length of the data buffers.
- ((ENH POINTER)) A double-word pointer to a buffer of enhancements to map to character cells in the vacated area.

First word = Buffer offset address
Second word = Buffer segment address

Display Enhancements Table

Security off:		@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Security on :		P	Q	R	S	T	U	V	W	X	Y	Z	[\]	-	

Half-Bright										X	X	X	X	X	X	X	X
Underline						X	X	X	X					X	X	X	X
Inverse Video				X	X			X	X			X	X			X	X
Blinking			X		X		X		X		X		X		X		X
No Enhancement		X															

If this pointer is null (segment=OFFFFH), then the vacated area's enhancements will not be altered.

- ((CHAR SET POINTER)) A double-word pointer to a buffer of character sets to map to character cells in the vacated area.

First word = Buffer offset address
Second word = Buffer segment address

Code	Character Set
@	Normal Roman
A	Line Drawing
B	Bold Face Roman
C	Italic Roman
D	Math
SPACE	The current character set

If this pointer is null (segment=OFFFHH), then the character sets in the vacated area will not be changed.

((CHAR POINTER))

A double-word pointer to a buffer of characters to map to the vacated area.

First word = Buffer offset address

Second word = Buffer segment address

If this pointer is null (segment=OFFFHH), the characters in the vacated area will be set to blanks.

(DIRECTION,DIST)

DIRECTION: The direction in which the data in the defined area is to be shifted.

0 = up

1 = down

2 = left

3 = right

DIST: The number of rows or columns to shift the current video data.

Outputs:

AX = 0

Successful

AX <> 0

Unsuccessful

Remarks:

Before you use this function, you must use the Defined Area function (0,1) to define a display area first.

This function does not change the position of the defined area. Just the data in the defined area is shifted, leaving a vacated area in the defined area.

The ASCII data in the defined area is shifted by a specified number of rows or columns. The vacated area is replaced with ASCII blanks. However, the enhancements and character sets are copied instead. As a result, you can choose to leave the data in the vacated area unchanged or you can assign new data to the vacated area. If you do not want to change the data in the vacated area, you will have to set ENH POINTER or CHAR SET POINTER to null (segment = OFFFHH).

Any data shifted off an edge of the defined area is lost. This function does not affect any data that is outside of the defined area.

If DATA LENGTH is greater than the number of character cells actually needed to fill the vacated area, this function uses only those bytes that are necessary to fill the vacated area. If DATA LENGTH is less, then only the number of character cells specified in DATA LENGTH will be changed.

AIOS Function Reference

Example:

This example assumes that the defined area has already been defined and data has already been written by Define Area and Write Area functions, respectively. (See examples in Define Area (0,1) and Write Area (0,2)). This example shifts the data in the define area one row down. The vacated area does not get updated.

Before the shift:

Characters		Enhancements		Character Sets	
10 11 12 13 14		10 11 12 13 14		10 11 12 13 14	
15	H E L L O	15	C C C C C	15	@ @ @ @ @
16	T H E R E	16	C C C C C	16	@ @ @ @ @

After the shift:

Characters		Enhancements		Character Sets	
10 11 12 13 14		10 11 12 13 14		10 11 12 13 14	
15	H E L L O	15	C C C C C	15	@ @ @ @ @
16	H E L L O	16	C C C C C	16	@ @ @ @ @

Note that after the shift, enhancements and character sets do not change in the vacated area (row 15); ASCII data is changed to blanks.

; Build the input parameter buffer with the specified input parameters.

```
IN_BUF  DW 6                ;function code
        DW 0                ;data buffer length
        DW OFFFFH,OFFFFH    ;enhancement pointer = null
        DW OFFFFH,OFFFFH    ;character set pointer = null
        DW OFFFFH,OFFFFH    ;character pointer = null
        DB 1                ;shift 1 row
        DB 1                ;direction = down
        .
        .
        .
; Set up registers to call AGIOS
; DS register is assumed to be the buffer's data segment
        MOV AX,4403H        ;I/O control write
        MOV BX,1            ;console handle
        MOV CX,18           ;input buffer length
        MOV DX,OFFSET IN_BUF ;buffer offset address
        INT 21H             ;call shift area
        RET                 ;return
```

See a Pascal-callable example of this function in Appendix C.

Write Line

This function writes a single row (or part of a row) anywhere in the display. The data is specified by three buffers. Each byte in a buffer corresponds to the respective display information for a character cell in the write line. Unlike the Write Area function, this intrinsic ignores the area bounds set by the Define Area function.

Input Buffer:

(0,7)	Function Code
(ROW,COLUMN)	Defines the row and column to display the first character. Row = 0 - 47 Column = 0 - 79
(DATA LENGTH)	Length of data buffers.
((ENH POINTER))	A double-word pointer to the buffer of enhancements to be written at the designated write line. First word = Buffer offset address Second word = Buffer segment address

Display Enhancements Table

Security off:		@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Security on :		P	Q	R	S	T	U	V	W	X	Y	Z	[\]	-	
Half-Bright												X	X	X	X	X	X
Underline						X	X	X	X					X	X	X	X
Inverse Video				X	X			X	X			X	X			X	X
Blinking			X		X			X		X			X		X		X
No Enhancement		X															

If this pointer is null (segment=OFFFFH), then the enhancements are not changed.

AIOS Function Reference

((CHAR SET POINTER)) A double-word pointer to a buffer of character sets to be written at the designated write line.

First word = Buffer offset address
Second word = Buffer segment address

<u>Code</u>	<u>Character Set</u>
@	Normal Roman
A	Line Drawing
B	Bold Face Roman
C	Italic Roman
D	Math
SPACE	The current character set

If this pointer is null (segment=0FFFFH), then the character sets are not changed.

((CHAR POINTER)) A double-word pointer to the buffer of characters to be written at the designated write line.

First word = Buffer offset address
Second word = Buffer segment address

If this pointer is null (segment=0FFFFH), then the characters are not changed.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function does not require the prior definition of an area.

If the position and length of the data violate the display boundaries, that portion of the data exceeding the boundary is ignored. No line wrap occurs.

If the requested row is not currently displayed on the screen at the time of the request, the Line function will still be performed. When that row is displayed, the written data will be visible.

Example:

See a Pascal-callable example of this function in Appendix C.

APPLICATION SOFTKEYS

The HP 150 system has three types of softkeys: system softkeys, user-defined softkeys, and application softkeys. System softkeys are maintained by the system. These keys allow access to terminal features; it is not possible for an application to monitor or use system softkeys. However most of the functionalities are accessible via escape sequences or via other AIOS functions. User-defined softkeys are intended to be used by the operator entering data directly from the keyboard. Although user-defined softkeys are not intended to be used by applications, many applications on the HP 3000 system use them for application purposes. The application softkeys on the HP 150 are designed to be used and interpreted by applications.

One thing you should note is that the softkeys (and touch labels) are active regardless of whether or not the labels are visible on the screen.

This section gives the AIOS function calls that support the application softkeys. The following application softkey functions are explained in detail in this section.

<u>Function Code</u>	<u>Function</u>
(0,8)	Update Softkey Label
(0,9)	Read Softkey Label
(0,10)	Not used
(0,11)	Display Softkey Labels
(0,12) - (0,15)	Not used

Application softkey functions in this section are used to manipulate softkey labels only. You should note that only the first eight application softkey labels can be displayed on the screen, the other four softkeys do not have labels.

The response string for an application softkey is a keycode. Keycodes 0 - B hex are designated for the 12 application softkeys, respectively. It is the application's responsibility to read the keycode, interpret the meaning of the keycode, and execute the keycode function accordingly.

In order to read the application softkeys, your application must be in keyboard intercept mode. That is, your application should turn on Raw mode (MS-DOS function 44H), turn on Keycode mode (AIOS function (0,43)), and define the application softkeys to be intercepted (AIOS function (0,40)). If your application is not in keyboard intercept mode, then the HP 150 system will ignore the keycode whenever an application softkey is pressed.

When in keyboard intercept mode, the keycode is returned to your application without any interpretation by the system. Your application should set up to read 4 bytes of information for each keyboard input. These 4 bytes give two bytes of key qualifier and two bytes of data. Key qualifier bytes and data bytes are explained in the Keycode mode section.

Update Softkey Label

This function changes an application softkey label and enhancement.

Input Buffer:

- (0,8) Function Code.
- (,NUMBER) Application softkey number (the softkey number is from 1 to 8 inclusive) (The high byte is a "don't care".)
- ((DATA)) A double-word pointer to a 16-byte buffer of ASCII data to be written into the label area. The first 8 bytes will be displayed on the top line; the next 8 bytes will be displayed on the bottom line.
- First word = Buffer offset address
 Second word = Buffer segment address
- (,TOP ENH) Enhancement code for the top half of the label. (The high byte is a "don't care".)
- (,BOT ENH) Enhancement code for the bottom half of the label. (The high byte is a "don't care".)

Display Enhancements Table

Security off:		@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Security on :		P	Q	R	S	T	U	V	W	X	Y	Z	[\]	-	

Half-Bright											X	X	X	X	X	X	X
Underline						X	X	X	X					X	X	X	X
Inverse Video				X	X			X	X			X	X			X	X
Blinking			X		X			X		X		X			X		X
No Enhancement		X															

Outputs:

- AX = 0 Successful
- AX <> 0 Unsuccessful

Remarks:

You can only change application softkey labels 1-8 because the other four softkeys do not have labels.

The double-word pointer, DATA, cannot be null. This function just updates the softkey labels; it does not display the labels if they are not already displayed.

To improve performance, you can use the batch function (0,0) to update all eight keys at once.

Read Softkey Label

This function gets the application softkey number specified by the caller and then returns the softkey label and the enhancement code in two buffers.

Input Buffer:

(0,9)	Function Code
(,NUMBER)	Application Softkey Number (The high byte is a "don't care".)
((DATA))	<p>A double-word pointer to a 16-byte buffer which will receive the ASCII data of the label.</p> <p>First word = Buffer offset address Second word = Buffer segment address</p> <p>If this pointer is null (segment=OFFFFH), then the label characters will not be returned.</p>
((ENHANCEMENTS))	<p>A double-word pointer to a 4-byte buffer which will receive the enhancement codes.</p> <p>First word = Buffer offset address Second word = Buffer segment address</p> <p>If this pointer is null (segment=OFFFFH), then the enhancements will not be returned.</p>

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful
DATA	The buffer has 16 bytes. The first 8 bytes are the top line label; the next 8 bytes are the bottom line label.
ENHANCEMENTS	<p>The buffer has 4 bytes:</p> <p>First byte = Top line enhancement code Second byte = Don't care Third byte = Bottom line enhancement code Fourth byte = Don't care</p>

Display Softkey Labels

This function displays the application softkey labels in the softkey window.

Input Buffer:

(0,11)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Once the softkey labels are displayed, pressing the function keys [f1] through [f12] generates the associated keycode. (Remember, only softkey labels 1 through 8 are visible when application softkeys are displayed; labels 9 through 12 are not visible.) However, you cannot assume that the softkeys are disabled even if they are not displayed.

Hidden softkeys are still "alive". That is, even though the labels are not visible on the screen, touching or pressing the function keys generates the keycode.

CONTROL FUNCTIONS

There are AIOS functions that let your application execute many of the terminal and display operations which are normally done with escape sequences. You may prefer to use these AIOS functions instead of sending escape sequences because these functions execute terminal operations much faster. In many cases, they are up to 5 times faster. However, if you are not programming with assembly language and terminal performance is not a concern, then sending escape sequences may be more desirable.

This section gives the AGIOS function calls that control the terminal of the HP 150 system. The following control functions are explained in detail in this section.

<u>Function Code</u>	<u>Function</u>
(0,16)	Execute Two-Character Escape Sequence
(0,17)	Position Cursor
(0,18)	Define Enhancements
(0,19)	Cursor Sense Absolute
(0,20)	Cursor Sense Relative
(0,21)	Set Cursor Type
(0,22) - (0,24)	Not Used
(0,25)	Read Fast AGIOS Entry Address
(0,26)	Go to Terminal Mode
(0,27) - (0,31)	Not Used

Your application can use function (0,16) to execute most of the two-character escape sequences supported on the HP 150 system.

The HP 150 system supports two cursor types, blinking underline or blinking inverse box. You can use function (0,21) to set to the preferred cursor type.

AIOS functions (0,17), (0,19), and (0,20) let you manipulate the cursor. Your application can sense the current cursor position as well as position it anywhere within the display. Your application can also use function (0,18) to define display enhancements. The HP 150 allows zero or more enhancements to be assigned to any character cell or any segment of a line on the display.

Execute Two-Character Sequence (ESC Char)

Input Buffer:

(0,16)

Function Code

OP-CHAR

The character equivalent of a 2-character escape sequence. Any operation characters are valid except for those that return data. The following list defines some of the most common ones.

0	Dump Alpha to Printer	U	Next
1	Set tab	V	Previous
2	Clear tab	W	Format Mode On
3	Clear all tabs	X	Format Mode Off
4	Set left margin	Y	Display Functions On
5	Set right margin	Z	Display Functions Off
9	Clear margins	[Start Unprotected Field
@	Delay one second]	End Unprotected Field
A	Cursor up	b	Unlock Keyboard
B	Cursor down	c	Lock Keyboard
C	Cursor right	g	Soft Reset Terminal
D	Cursor left	h	Home Up
E	Terminal hard reset	i	Back tab
F	Home down	j	Display User-Defined Softkey
G	Return		Definition Menu
H	Home up	k	Exit User-Defined Softkey
I	Tab		Definition Menu
J	Clear display	l	Memory Lock On
K	Clear line	m	Memory Lock Off
L	Insert one line		
M	Delete one line		
N	Insert character with wraparound		
O	Delete character with wraparound		
P	Delete character without wraparound		
Q	Insert character without wraparound		
R	Insert character off		
S	Scroll up		
T	Scroll down		

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function generally executes faster than sending escape sequences.

Position Cursor (ESC & a)

This function moves the cursor to a specified row and column depending on a given mode.

Input Buffer:

(0,17)	Function Code
MODE	1 Byte
	Bit 7 0 ----- -----
Bit 0:	1 = Screen row address (0-23) 0 = Display row address (0-47)
Bit 1:	1 = Row address (relative to the current cursor position) 0 = Absolute row address
Bit 2:	1 = Negative row address (only meaningful if bit 1 = 1) 0 = Positive row address
Bit 3:	1 = Use given ROW to calculate new row address 0 = Do not change row address
Bit 4:	1 = Screen column address (0-79) 0 = Display column address (0-79)
Bit 5:	1 = Relative column address (relative to current cursor position) 0 = Absolute column address
Bit 6:	1 = Negative column address (only meaningful if bit 5 = 1) 0 = Positive column address
Bit 7:	1 = Use given COLUMN to calculate new column address 0 = Do not change column address
(COLUMN)	A positive integer indicating the new column number (0-79) or the number of columns to move the cursor if bit 5 of the MODE parameter is set.
(ROW)	A positive integer indicating the new ROW number (0-47) or the number of rows to move the cursor if bit 1 of the MODE parameter is set.

AIOS Function Reference

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Column and row parameters do not have to be both absolute or relative; they do not have to be both screen addresses or both display addresses. If positive or negative bit is set for an absolute address, the bit will be ignored.

Row and column addresses do not both have to be changed. That is, you can choose to move the cursor to a different row without changing the column position, or vice versa.

Example:

This example has a program segment that calls function (0,17) to move the cursor to column 10 and row 5.

```
;
;Build input buffer to call AIOS.
;
IN_BUF  DW 17                ;position cursor function code
        DB 88H               ;MODE - gives row and column
        DW 9                 ;move to COLUMN 10
        DW 4                 ;move to ROW 5
        .
        .
        .
;Set up registers to call AIOS
;DS register is assumed to be the current data segment
MOV AX,4403H                ;I/O control write
MOV BX,1                    ;console handle
MOV CX,7                    ;7-byte buffer
MOV DX,OFFSET IN_BUF        ;buffer offset
INT 21H                     ;MS-DOS call to move the cursor
RET                          ;return
```

Define Enhancements (ESC & d).

Input Buffer:

(0,18)	Function Code
SECURITY	1 byte
	10H = ON
	This means that characters are <u>not</u> displayed on the screen.
	0H = OFF
	This means that characters are displayed on the screen.

ENHANCEMENT	1 byte
	An enhancement code from '@' (40H) through 'O' (4FH).

Display Enhancements Table

Enhancements:	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Half-Bright									X	X	X	X	X	X	X	X
Underline					X	X	X	X					X	X	X	X
Inverse Video			X	X			X	X			X	X			X	X
Blinking		X		X		X		X		X		X		X		X
No Enhancement	X															

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The specified enhancement code applies to characters at and to the right of the current cursor position. The enhancement remains active until a NO enhancement code (@) is specified, the end of the cursor line is reached, or the cursor is moved to another line.

Cursor Sense Absolute (ESC a)

This function returns the current cursor's absolute display column and row.

Input Buffer:

(0,19)	Function Code
((BUFFER))	A double-word pointer to a buffer where two words are returned. The first word is the column number and the second word is the row number.
	First word = Buffer offset address
	Second word = Buffer segment address

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful
(BUFFER)	First word = Absolute display column (0-79)
	Second word = Absolute display row (0-47)

Cursor Sense Relative (ESC `)

This function returns the current cursor's relative screen column and row.

Input Buffer:

(0,20)

Function Code

((BUFFER))

A double-word pointer to a buffer where two words are returned. The first word is the column number and the second word is the row number.

First word = Buffer offset address

Second word = Buffer segment address

Outputs:

AX = 0

Successful

AX <> 0

Unsuccessful

(BUFFER)

First word = Screen column (0-79)

Second word = Screen row (0-23)

Set Cursor Type

This function sets the alphanumeric cursor type.

Input Buffer:

(0,21)	Function Code
(,TYPE)	Alpha cursor type: 0 = Blinking underscore 1 = Blinking inverse box
	(The high byte is a "don't care".)

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function just changes the physical appearance of the cursor; it does not change the cursor type field in the HP 150 Global Configuration Menu.

Read Fast AGIOS Entry Address

This function returns the segment and offset address of the entry point for executing AGIOS functions. Calling this returned address executes AGIOS functions faster than using MS-DOS function 44 hex because the MS-DOS function 'overhead' is eliminated. In order to call the returned address, your application has to set up the same input buffer as required by function 44 hex, and the following registers:

CX = Number of bytes in the input buffer
 DX = Input buffer's offset address
 DS = Input buffer's segment address

Input Buffer:

(0,25) Function Code
 ((BUFFER)) A double-word pointer to a 2-word buffer where the entry address will be returned.

Outputs:

AX = 0 Successful
 AX<> 0 Unsuccessful
 (BUFFER)
 First word = AGIOS entry offset address
 Second word = AGIOS entry segment address

Remarks:

To call AGIOS using the returned address, AX and BX registers are not required. Upon return from the fast call, the contents in AX, SP, and segment registers remain unchanged; the other registers are destroyed.

NOTE

This function is included in Operating System version B.01.00 or later.

Go to Terminal Mode

This function lets your application switch to Terminal mode programmatically. A user can return to the application (Computer mode) by pressing [Shift][Stop] keys from the keyboard.

Input Buffer:

(0,26)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function operates the same as pressing the "Terminal" softkey from P.A.M.

NOTE

This function is included in Operating System version B.01.00 or later.

TOUCHSCREEN FUNCTIONS (ESC - z)

The touch features on the HP 150 can be programmed in a variety of ways. The two general types of touch operations are "Field" operations and "Row/Column" operations. These two types can be intermixed.

Several of the touchscreen function calls in this section assume Keycode mode. (Refer to the section on Keycode Mode for information on this mode.)

This section gives the AIOS function calls that support the touchscreen. The following touchscreen functions are explained in detail in this section.

<u>Function Code</u>	<u>Function</u>
(0,32)	Define Touch Field
(0,33)	Define Softkey Field
(0,34)	Delete Touch Field
(0,35)	Touchscreen Reset
(0,36)	Set Touch Reporting Modes
(0,37) - (0,39)	Not Used

Field Operations

There are four types of touch fields you can define. They are:

ASCH FIELDS. This mode is very similar to the user-definable softkeys. A buffer of characters is associated with a touch field. A response string of 0 to 80 ASCII characters is obtained by consecutive keyboard input operations. The first input obtains the first ASCII byte, and the second input obtains the second ASCII byte, etc. The response string is generated when the field is touched and should be indistinguishable from the typing of the same string from the keyboard. When Keycode mode is on, the qualifier word of each data byte returned to the application has the touchscreen ID (OFFH). Auto-repeat is performed.

KEYCODE FIELDS. Keycode fields require that the terminal is in Keycode mode. There are two words in the response string; they are a qualifier word and a keycode word. The qualifier word has the keyboard ID (OCOH). Auto-repeat is performed. This type of field is not supported when defined with the ESC - z sequence.

TOGGLE FIELDS. The touch field is defined as a toggle switch. Touching the area toggles the field on and off. Whenever the field is touched, sensing information is passed to the application. The sensing information consists of three data bytes. The first byte is the touch status (ON or OFF). The other two bytes are the defined response string; they are usually used to

AIOS Function Reference

identify the field. The data is obtained by three consecutive keyboard input operations. If Keycode mode is on, the qualifier word of each data byte returned to the application has the touchscreen ID (80 hex). The three data bytes for toggle on field report are:

- 01H - Toggle on field report code
- D1 - Response string first byte
- D2 - Response string second byte

The three data bytes for toggle off field report are:

- 02H - Toggle off field report code
- D1 - Response string first byte
- D2 - Response string second byte

NORMAL FIELDS. This type of touch field senses touch and/or release. The sensing information consists of three data bytes. The first byte is the touch status (touch or release). The next two bytes are the defined response string; they are usually used to identify the field. The data can be obtained by three consecutive keyboard input operations. When Keycode mode is on, the qualifier word of each data byte returned to the application has the touchscreen ID (80 hex). Auto-repeat is performed. The three data bytes for field touched report are:

- 05H - Field touched report code
- D1 - Response string first byte
- D2 - Response string second byte

The three data bytes for field released report are:

- 06H - Field released report code
- D1 - Response string first byte
- D2 - Response string second byte

Again, the main difference between toggle fields and normal fields is that toggle fields have two states, ON and OFF. A normal field is ON only while it is actually touched.

Row/Column Operations

This type of touch operation returns the row and column position when a touch occurs. The row and column position are returned byte by byte using the keyboard input function of the operating system. Three data bytes are returned. The first byte is the touch status (touch or release). The second byte is the touched row number, and the third byte is the touched column number. If Keycode mode is on, then the qualifier word returned with each byte of data has the touchscreen ID (080 hex). The three data bytes for touched row/column report are:

03H - Row column touch report code
Row - Touched row number in binary
Col - Touched column number in binary

The three data bytes for released row/column report are:

04H - Row column release report code
Row - Touched row number in binary
Col - Touched col number in binary

General Procedure for Using Touchscreen

The following lists a recommended procedure for programming the touchscreen. Steps 1-5 should always be done; they may be executed all at once using the AGIOS Batch Command, function (0,0). If your application needs to control keyboard and touchscreen inputs, then steps 6-8 should also be included.

1. Delete all fields using function (0,34)
2. Enable softkey reporting for all 8 softkeys using function (0,33)
3. Select touchscreen reporting mode, using function (0,36)
4. Define fields, if any, using function (0,32)
5. Turn on touchscreen using function (0,36)
6. If necessary, turn on Keycode mode using function (0,43)
7. Define key characteristics using function (0,40)
8. Put console device in Raw mode using MS-DOS function 44 hex (MS-DOS function 44 hex is discussed in the *Microsoft Programmer's Reference Manual*.)

Define Touch Field (ESC - z g)

This function lets you define a touch field.

Input Buffer:

(0,32)	Function Code
((STRING))	A double-word pointer to the buffer which stores the response string.
	Keycode field: (2 words) First word = Qualifier Second word = Keycode
	Toggle or Normal Field: (2 bytes) ASCII data
	ASCII field: (0-80 bytes) ASCII data
(LENGTH)	Number of bytes in the response string
(ATTRIBUTE,MODE)	Touch ATTRIBUTE: 1 = ASCII field 2 = Keycode field (not supported when defined with escape sequence.) 3 = Toggle field 4 = Normal Field
	Reporting MODE (only valid for ATTRIBUTE 1 and 4 above) 1 = Report when touched 2 = Report when released 3 = Report both touch and release

AIOS Function Reference

(ON-ENH,OFF-ENH) The ON and OFF enhancements of toggle fields. For ASCII, keycode and normal fields, ON-ENH is the enhancement when the field is touched; OFF-ENH is the enhancement when the field is not touched.

Display Enhancements Table

Security off:		@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Security on :		P	Q	R	S	T	U	V	W	X	Y	Z	[\]	-	

Half-Bright										X	X	X	X	X	X	X	X
Underline						X	X	X	X					X	X	X	X
Inverse Video				X	X			X	X			X	X			X	X
Blinking			X		X			X		X		X			X		X
No Enhancement		X															

(CURSOR,BEEP) CURSOR:
 0 = Do not change the cursor position
 1 = When touch, position the cursor
 to the upper left corner of the field

BEEP:
 0 = Do not beep
 1 = Beep on touch

(LR-ROW,LR-COL) Row and column of the lower right corner of the touch field; row and column are display relative.

LR-ROW = 0-47
LR-COL = 0-79

(UL-ROW,UL-COL) Row and column of the upper left corner of the touch field; row and column are display relative.

UL-ROW = 0-47
UL-COL = 0-79

Remarks:

To define a Keycode field, you must supply a valid keycode from the Keycode Table that appears in the Keyboard Processing section. Furthermore, the system always returns the Keyboard Device ID (0C0H) for Keycode fields, regardless of what you supply in the Qualifier word. (See the section on Keycode mode for more information.)

Appendix B has an example on defining touch fields from BASIC.

Define Softkey Field (ESC - z s)

This function defines one of the eight softkey label areas as a touch field. These fields when touched produce the same response as if the corresponding function key is pressed. The default is all softkey touch fields are on.

Input Buffer:

(0,33)

Function Code

(MODE,KEY)

KEY (Softkey number): 1-8

MODE: 1 = softkey label is touch sensitive

0 = softkey label is not touch sensitive

Outputs:

AX = 0

Successful

AX <> 0

Unsuccessful

Remarks:

In general, softkey touch field should not be turned off.

Delete Touch Field (ESC - z d).

Deletes the touch field with upper left corner at the given row and column. The row and column are display relative coordinates.

Input Buffer:

(0,34)	Function Code
(UL-ROW,UL-COL)	Row and column position in the field to be deleted.
	UL-ROW = 0-47
	UL-COL = 0-79
	When UL-ROW = OFFH and UL-COL = OFFH, all fields will be deleted.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Actually, parameters UL-ROW and UL-COL do not have to be the upper left corner of the field. This function deletes a field as long as UL-ROW and UL-COL specify a position within the desired field.

If overlapping fields are defined, then the most recently created field will be deleted.

If there is no touch field at the given row and column position, nothing happens.

During initialization, your application should delete all touch fields that may be defined by an earlier application executed on the system.

Touchscreen Reset (ESC - z j)

Turn all touch fields to the OFF state

Input Buffer:

(0,35)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function does not delete any touch fields. It just turns all touch fields off; all fields are displayed with OFF enhancements.

Set Touch Reporting Modes (ESC - z n)

This function determines if, and how, touchscreen reporting is handled by the HP 150.

Input Buffer:

(0,36)	Function Code
(,SCREEN-MODE)	Touch Field and Row/Column sensing: (The high byte is a "don't care".)
0 -	Disable reporting. This has the effect of turning off reporting without deleting any fields.
1 -	Enable sensing for row/column position. Touch fields are inactive.
2 -	Enable sensing for touch fields only. Row/column sensing is inactive.
3 -	Enable sensing for both row/column and touch fields. Row/column sensing occurs for areas not defined as touch fields.
4 -	Toggles touchscreen on and off. When OFF, all touchscreen operations are disabled. This disables softkey fields as well. The start-up default value is ON.
10 - 14	Same as 0-4, but causes escape sequence reports to be sent in the following form.

ESC - z <string> <type>Q <Carriage Return>

with <type> :

- 1 = Toggle field turning on report, <string> is the two-character response string.
- 2 = Toggle field turning off report, <string> is the two-character response string.
- 3 = Row/Column touch report, <string> is <row> x <column> y.
- 4 = Row/Column release report, <string> is <row> x <column> y.
- 5 = Normal field touch report, <string> is the two-character response string.

6 = Normal field release report, <string>
is the two-character response string.

(,TOUCH-MODE)

Sense touch or release: (used with
Row/Column sensing only; the high
byte is a "don't care")

- 1 - Report on Touch
- 2 - Report on Release
- 3 - Report on both Touch and Release

Outputs:

AX = 0
AX <> 0

Successful
Unsuccessful

Remarks:

If an application wishes to disable touchscreen, it should set the SCREEN-MODE
parameter to '0'.

KEYBOARD PROCESSING

Keyboard Processing functions let you gain more control over the use of the keyboard and touchscreen. This section gives the AIOS function calls that support keyboard processing. The following keyboard processing functions are explained in detail in this section.

<u>Function Code</u>	<u>Function</u>
(0,40)	Define Key Characteristics
(0,41)	Get Key Characteristics
(0,42)	Put Key

The keyboard consists of two types of keys:

- o The ASCII keys (letters, digits, control codes, and punctuation marks).
- o Special keys which can be divided into two classes:
 - numeric/graphics keypad keys
 - other special function keys (Insert Line, Insert Character, etc.)

There is a keycode assigned to each special function key on the keyboard. These keycodes are listed in the Keycode table in this section. It should be noted that the keycodes for [f1] through [f12] are valid only when the application softkey labels are enabled with AIOS function call (0,11).

Key Characteristics

Using the Keyboard Processing functions, each of the special keyboard functions can be individually set to normal processing or one of the special processing modes (beep, intercept, or ignore).

NORMAL. This is the default action where the system processes the key.

INTERCEPT. The key is passed to the application; no action is performed by the system.

IGNORE. No system action is performed and no report is generated.

BEEP. The system generates a 'beep' when a key is pressed in combination with the above characteristics.

In order to gain total control of reading keys from the keyboard or from the touchscreen, your application should be in keyboard intercept mode. That is,

your application should *first define each of the special keys on the keyboard to the desired mode of operation, put the operating system's console input device into Raw mode, and turn on Keycode mode.*

You can use AIOS function (0,40) to define each special function key to the desired mode. You can choose to process the key normally, intercept the key, or ignore the key. You will probably notice that the ASCII displayable characters are not in the Keycode table that appears later in this section. This is because whenever Keycode mode and Raw mode are ON, the ASCII keys are intercepted automatically.

Whenever the operating system's console input device is in Raw mode, all those keyboard functions that are normally processed by the operating system will be passed to your application without any interpretation by the operating system. You should use the MS-DOS function 44 hex to put the console input device into Raw mode. The default is Raw mode OFF (see the *Microsoft Programmer's Reference Manual* for more information on function 44 hex.) There is also a sample program segment on turning Raw mode on and off in the next section on Keycode mode.

Whenever Keycode mode is ON, your application will receive 4 bytes of information for each key stroke, two bytes of Data and two bytes of Qualifiers. You should use AIOS function (0,43) to turn on Keycode mode. (See the next section on Keycode Mode for more information on reading keycodes.)

As a result, to properly use keyboard processing functions to control the keyboard and touchscreen, your application should:

1. Define special function key characteristics to either process, intercept, or ignore keys. (Use AIOS function (0,40))
2. Set the console input device into Raw mode so that keys are not executed by the operating system. (Use MS-DOS function 44 hex)
3. Turn on Keycode mode to examine the key attributes. (Use AIOS function (0,43))

Note that if any of the above three modes is not set up correctly, then your application will not be able to gain total control of the keyboard and the touchscreen.

Once your application is set up to control the keyboard and touchscreen, there is no explicit "get keycode and qualifier" function call. The standard operating system console input function returns the normal ASCII code and also keycode. The qualifiers are also returned if Keycode mode is ON. If Keycode mode is ON, you may use MS-DOS function call 3F hex to read 4 bytes of data for each keypress.

After your application examines the pressed key, it can either process it specially by the application or process it normally by the system. To process a key normally after it has been intercepted in Keycode mode and Raw mode, use the AIOS Put Key function (0,42).

Special Keyboard Keys Definition

The following section describes how the HP 150 system interprets some of the special keys on the keyboard. An application can intercept, inhibit, or alter the meaning of some of these keys.

[Break] and [Reset] Keys

[Break]

This key sends a data communications break signal to the remote computer.

[Shift] [Break]

This key causes a soft-reset of the HP 150 system. An application cannot intercept this key.

[CTRL] [Shift] [Break]

This key causes a hard-reset of the HP 150 system. An application cannot intercept this key.

[Stop] Key

[Stop]

This key toggles between starting and stopping the system. It causes the HP 150 to enter a loop waiting for a [Stop] or a hard-reset keypress.

[Shift] [Stop]

In Terminal mode, the HP 150 will attempt to load the operating system.

[CTRL] [Stop]

This key functions identically to [Shift] [Stop]. However, an application cannot intercept this key.

[Enter] Key

[Enter]

In Block mode, this key can be strapped for line or page transmission using no handshaking, short handshaking (DC1), or long handshaking (DC1/DC2/DC1). You can select the Return key to act as if it were the Enter key in the Terminal Configuration menu.

[Shift] [Enter]

This key sends the entire alphanumeric display data to the currently selected print device.

[CTRL] [Enter]

The key does a raster dump of the graphics display memory to the currently selected print device.

[Menu] Key

[Menu]

This key toggles the softkey labels on and off. If softkey labels are toggled off, you can press [User System], [Shift] [User System], or [Ctrl] [User System] to display the softkey labels again. You should remember that softkeys are still active even when their labels are hidden.

[Shift] [Menu]

This key does not perform any function in the HP 150 system, but it is interceptable.

[CTRL] [Menu]

This key displays the User-Definable Softkey Menu. An application cannot intercept this key.

[CTRL] [Shift] [Menu]

This key toggles the entire touchscreen on or off. An application cannot intercept this key.

[User System] Key

[User System]

This key displays the current level of system softkeys. Pressing [User System] a second time always displays the root level of the system softkeys.

[Shift] [User System]

This key displays the application softkeys. An application cannot intercept this key.

[CTRL] [User System]

This key displays the User-Defined softkeys. An application cannot intercept this key.

Cursor Movement Key

[Select]

An application can use this key to let a user select the item at which the cursor is pointing. If this key is not intercepted, it sends an "ESC & p" sequence, which does not perform any action in the HP 150.

Local Editing Keys

[Shift] [Clear Line]

This key clears the current cursor line regardless of current cursor position.

[Shift] [Clear Display]

This key clears the entire display memory regardless of current cursor position.

AIOS Function Reference

Alphanumeric Scroll Key

[Next]

This key moves the display memory text up in the screen so the line immediately following the current last screen line becomes the new first screen line. [Next] always leaves at least one defined line (possibly a blank line) in the screen. [Next] positions the cursor in Row 1 and Column 1 at the end of this operation.

[Prev]

This key moves the display memory text down in the screen so the line immediately preceeding the current first screen line becomes the new last screen line. If there are not enough lines to fill the screen, [Prev] places the first line of display memory at the top of the screen. [Prev] positions the cursor in Row 1 and column 1 at the end of this operation.

Numeric/Graphics Keypad

[CTRL] [Keypad "-"]

This key toggles between Numeric Pad and Graphics Pad modes.

In Graphics Pad mode, the keypad keys return the same keycodes or perform the same functions as the corresponding graphics function keys on the main keyboard.

In Numeric Pad mode without interception, the keypad keys return ASCII codes corresponding to the characters on the keycaps.

In Numeric Pad mode with interception, the keypad keys return unique keycodes (e.g., the numeric-pad zero key is different from the main-keyboard zero key), which are also independent of the Shift key.

You should note that pressing the Shift key along with a numeric-pad key displays a different character on the screen, but the Keycode returned is independent of the Shift Key. Therefore, your application should examine the 'Special' and 'Shift' bits in the Qualifier Word to distinguish the various permutations.

The shifted Numeric Pad keys are mapped as follows:

<u>Numeric Keypad Keys</u>	<u>Equivalent Character</u>
[Shift] [.]	Tilde "~"
[Shift] [0]	Caret " ^ "
[Shift] [1]	Left Brace "{ "
[Shift] [2]	Vertical Bar " "
[Shift] [3]	Right Brace "} "
[Shift] [4]	Left Bracket "["
[Shift] [5]	Backslash "\ "
[Shift] [6]	Right Bracket "] "
[Shift] [7]	Cross Hatch "# "
[Shift] [8]	Back Quote "` "
[Shift] [9]	Commercial At "@ "

Keycode Table

<u>Key(s)</u>	<u>Keycode</u>	<u>Key(s)</u>	<u>Keycode</u>
f1	00H	Insert-Line	44H
f2	01H	Delete-Line	45H
f3	02H	Insert-Char with wraparound	46H
f4	03H	Delete-Char with wraparound	47H
f6	05H	Shift/Clear-display	48H
f7	06H	Shift/Clear-line	49H
f8	07H		
f9	08H	Enter	50H
f10	09H	User System	51H
f11	0AH	Select	52H
f12	0BH	Menu	53H
		Break	54H
Cursor-Up	20H	ESC	55H
Cursor-Down	21H	CAPS	56H
Cursor-Right	22H	DEL	57H
Cursor-Left	23H	Stop	58H
		Shift/Menu	59H
Tab	24H	Shift/Stop	5AH
Return	25H	Shift/Enter	5BH
Back-tab	26H	CTRL/Enter	5CH
Back-space	27H		
Scroll-Down	28H	Graphics Pad:	
-Up	29H	Cursor-Left	61H
-Left	2AH	Cursor-Down	62H
-Right	2BH	Cursor-Right	63H
		Roll-Up	64H
Home-Up	2CH	Cursor-Up	65H
Home-Down	2DH	Roll-Left	66H
		Roll-Down	67H
Next	2EH	Roll-Right	68H
Prev	2FH	Clear Display	69H
		Pad-Toggle	6AH
Insert-Char	40H	Alpha-Display	6BH
Delete-Char	41H	Graphics-Cursor	6CH
Clear-Display	42H	Graphics Display	6DH
Clear-Line	43H	Home-Up	6EH
		Home-Down	6FH

***** There are more keycodes listed on the next page *****

<u>Key(s)</u>	<u>Keycode</u>
Numeric Pad:	
Zero	30H
One	31H
Two	32H
Three	33H
Four	34H
Five	35H
Six	36H
Seven	37H
Eight	38H
Nine	39H
Minus	70H
Asterisk	71H
Plus	72H
Slash	73H
Comma	74H
Enter	75H
Tab	76H
Period	77H

NOTE

Keycodes 0CH - 1FH and 4AH - 4FH are not used.

Keycode Notes

1. Function keys f1-f12 and the numeric-pad keys return the same keycodes independent of [Shift] and [CTRL] key operation. Your application can examine the Qualifier word to distinguish the various permutations.
2. Applications cannot intercept other functions that are not listed in the Keycode table. The non-interceptable functions are listed below:
 - Soft Reset ([Shift] [Break])
 - Hard Reset ([CTRL] [Shift] [Break])
 - Abort ([CTRL] [Stop])
 - User-Definable Softkey Menu ([CTRL] [Menu])
 - Touchscreen On/Off ([CTRL] [Shift] [Menu])
 - Application Softkeys ([Shift] [User System])
 - User-Defined Softkeys ([CTRL] [User System])
3. If [CTRL] and [Shift] are both depressed with a key for which the combination is not defined, the keycode returned will be the same as if only [CTRL] were pressed. However, the Qualifier word will indicate both qualifiers, [CTRL] and [Shift]

(See the section on Reading Keycodes for additional information on Qualifier word.)

Define Key Characteristics

This function lets you alter characteristics of any of the Special keys, not the ASCII keys. Specifically you can:

- o Process the key normally
- o Intercept the key and pass a keycode to the application for processing
- o Ignore the key when it is pressed
- o Beep when the key is pressed in combination with the above characteristics

Input Buffer:

(0,40)	Function Code
(CHARACTERISTICS)	Key Characteristics

Bit 15		2	1	0

<-----Reserved----->				

Bit 0 = 1 => Beep
 Bit 1 = 1 => Intercept
 Bit 2 = 1 => Ignore
 3-15 Reserved (should be set to zero)

If bits 1 and 2 are both set, the key is treated as an intercept key. When both are zero, the key resumes normal functioning.

(KEYCODE) Keycode corresponds to the Special key on the keyboard to be defined. A value of OFEH will set all Special keys to the specified characteristics. (See the Keycode Table listed earlier for valid keycodes.)

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

***** See an example of this function on the next page *****

AIOS Function Reference

Example:

The following example defines all keys to be intercepted.

```
;
;Build the input buffer to call AIOS function (0,40).
;
IN_BUF    DW    40                ;define key characteristics
          DW    2                ;set intercept mode
          DW    OFEH             ;keycode = all keys
          .
          .
          .
;
;Set up registers to call AIOS function.
;DS is assumed to be the buffer's data segment.
;
      MOV  AX,4403H              ;I/O control write
      MOV  BX,1                  ;console handle
      MOV  CX,6                  ;6 bytes in input buffer
      MOV  DX, OFFSET IN_BUF     ;input buffer offset address
      INT  21H                  ;call AGIOS
      RET                       ;return
```

Get Key Characteristics

This function returns the characteristics of a Special key.

Input Buffer:

 $(0, 41)$

Function Code

((BUFFER))

A double-word pointer to a 1-word buffer where the key's characteristics will be returned.

First word = Buffer offset address

Second word = Buffer segment address

(KEYCODE)

The keycode corresponds to the desired Special key (see Keycode Table listed earlier for valid keycodes).

Outputs:

$$AX = 0$$

Successful

AX <> 0

Successful
Unsuccessful

(BUFFER)

The characteristic word is defined as follows:

Bit 15		2	1	0

	<-----Reserved----->			

Bit 0 = 1 => Beep

Bit 1 = 1 => Intercept

Bit 2 = 1 => Ignore

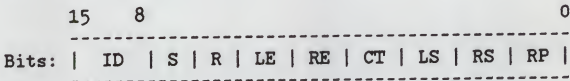
Put Key

This function lets you direct the system to process the keycode normally. For example, an application can intercept a keypress and then use this function to tell the system to process the key normally.

Input Buffer:

(0,42) Function Code

(QUALIFIER) Qualifier



Bit Interpretation (1=set, 0=not set)

- 15-8 = Input Device ID
(should be set to the keyboard ID = 0COH)
- 7 = Special key (must be set).
- 6 = Reserved
- 5 = Left extend char, set when down
- 4 = Right extend char, set when down
- 3 = Control, set when down
- 2 = Left shift, set when down
- 1 = Right shift, set when down
- 0 = Not used

(KEYCODE) The keycode corresponds to the desired Special key.
 (See Keycode Table listed earlier for valid keycodes).

Outputs:

AX = 0 Successful

AX <> 0 Unsuccessful

Remarks:

This function can be used to simulate keyboard pressing of a Special key. Therefore, the high byte of the parameter QUALIFIER should be set to 0COH for keyboard input. (See the section on Reading Keycodes presented later for more information.)

For normal ASCII keys, you would simply print (or 'echo') the character in place of using this function.

KEYCODE MODE

Keycode mode allows applications to obtain additional control over the HP 150 keyboard. By using this mode, your application can detect keypresses as well as the state of the five 'shift-type' keys: left and right [Shift] key; left and right [Extend char] keys; and the [CTRL] key. It also lets you distinguish between touchscreen and keyboard input.

To obtain this control requires no special interrupts or calls; all keycode activity is directed to standard console input.

This section gives the AIOs function calls that support Keycode mode. It also lists the function that reads the Graphics/Numeric keypad status. The following functions are explained in detail in this section.

<u>Function Code</u>	<u>Function</u>
(0,43)	Turn Keycode Mode On/Off
(0,44)	Read Keycode Mode Status
(0,45)	Read Numeric/Graphics Keypad Status
(0,46)	Read Key

Using Keycode Mode

Keycode mode is normally used with Raw mode and keyboard intercept. To properly use Keycode mode, your application will need to do the following set-up:

- o Define Key Characteristics with AIOs call (0,40)
- o Turn on Raw Mode with MS-DOS call 44 hex
- o Turn off Control-C checking with MS-DOS call 33 hex
- o Turn on Keycode mode with AIOs call (0,43)

When all this has been done, your application will need to read four bytes of data for each keypress; more will be said on this shortly (see Reading Keycodes).

Defining Special Keys

Key characteristics can be specified for any of the Special function keys on the HP 150. Key characteristics include the ability to process a key normally, to intercept a key, or to ignore it. When in Keycode mode, each key characteristic is affected in the following way:

NORMAL. This is the default action; the system processes the key.

INTERCEPT. The system generates 4 bytes for each key pressed. When in this mode, your application should be set up to read 4 bytes per key input.

IGNORE. No system action is performed and no report is generated.

You can also have the HP 150 'beep' when individual keys are pressed, in any combination with the other key characteristics. (See AIOS function (0,40) for additional information on Define Key characteristics.)

Using Raw Mode

MS-DOS has two processing modes: Cooked mode and Raw mode. In Cooked mode, the default mode for devices, MS-DOS checks each character for special significance. For example, Tab, Back Space, Control-P, and Control-Z all have special meanings to MS-DOS. When a device is in Raw mode, MS-DOS stops checking for these special characters. Keys are passed to your application without any interpretation by MS-DOS. Raw mode generally allows faster operation on devices.

The following sample assembly routines turn Raw mode on and off.

```

RAWON:  MOV AX,4400H      ;Get device information
        MOV BX,1         ;File handle = 1 for console
        INT 21H
        XOR DH,DH        ;Clear high byte for put
        OR  DL,20H       ;Set raw bit without changing other bits
        MOV AX,4401H     ;Put device information
        MOV BX,1         ;File handle = 1 for console
        INT 21H         ;Turn on Raw mode
        RET              ;Done = return to caller

RAWOFF: MOV AX,4400H     ;Get device information
        MOV BX,1         ;File handle = 1 for console
        INT 21H
        XOR DH,DH       ;Clear high byte for put
        AND DL,0DFH     ;Clear raw bit without changing other bits
        MOV AX,4401H     ;Put device information
        MOV BX,1         ;File handle = 1 for console
        INT 21H         ;Turn off Raw mode
        RET              ;Return to caller

```

Control-C trapping is normally handled by MS-DOS when I/O function calls occur. You want to verify that this is disabled by setting Control-C trapping off with MS-DOS function 33 hex.

Turning On Keycode Mode

Finally, turn on Keycode mode with AGIOS call (0,43).

```
SETKC: MOV  AX,4403H      ;IO Ctrl Write
        MOV  BX,1         ;Console handle
        MOV  CX,3         ;Buf length
        MOV  DX,OFFSET BUF
        INT  21H
        RET
BUF      DB  43           ;Keycode mode function number
        DB  0             ;AIOS index
        DB  1             ;Keycode mode on
```

Before your application ends, be sure to 'clean up'. This usually means setting all keys to their default characteristics, turning off Keycode mode, and setting Cooked mode on the console. This is good programming practice and means the next program to run will not have problems because of your application.

Reading Keycodes

Once Keycode mode is enabled, each keypress generates four bytes: Device ID, Shift Bits, Null, and Data. Device ID and Shift Bits bytes together are called the Qualifier word, and Null and Data bytes together are called the Data word. You should note that if you are reading these four bytes of information in byte mode, they are listed in the following order:

```
Shift Bits  <----- first byte
Device ID
Data
Null        <----- last byte
```

However, in word mode, then they are stored in pairs of 16 bits:

```
(Device ID, Shift Bits)
(Null,          Data)
```

QUALIFIER WORD The two-byte qualifier word is in the following format:

```
      High Byte      Low Byte
+-----+-----+
| Device ID | Shift Bits |
+-----+-----+
```

The high order byte (Device ID) field indicates which device was the source of the key. Possible values for Device ID are:

<u>Device ID</u>	<u>Data Source</u>
00 Hex	Internal Code
080 Hex	Touchscreen (Keycode, Toggle, and Normal fields)
0C0 Hex	Keyboard
0FF Hex	Touchscreen (ASCII field)

Internal Code is used when the system responds to an inquiry (such as the default response string of the user-defined softkeys).

NOTE

If you select a softkey function by touching the softkey label on the screen, or if you touch a Keycode field defined on the screen, the system still returns the Keyboard Device ID (0C0H) in the Device ID byte.

AIOS Function Reference

The low order byte (Shift Bits) field indicates additional information about the keypress. You can determine which, if any, shift keys were depressed; or whether the key is an ASCII key or a Special key; or whether this keypress is due to auto-repeating on the keyboard. The format of the Shift Bits byte is listed below:

Bits	7								0								
	+-----+																
		S	:	R	:	LE	:	RE	:	CT	:	LS	:	RS	:	RP	
	+-----+																

The meaning of each bit follows:

- S: Special Bit. This bit indicates the key is one of the 'Special' keys and the Data Word contains a keycode from the Keycode Table listed in the Keyboard Processing Section. When this bit is cleared, the Data Word contains a Roman-8 ASCII character code.
- R: Reserved Bit. This bit is reserved for future expansion.
- LE: These bits indicate the state of the left (LE) and right (RE) [Extend char] key. These keys normally shift into additional 8 bit ASCII characters.
- RE:
- CT: This bit indicates the control key [CTRL] was pressed.
- LS: These bits indicate the state of the left (LS) and right (RS) shift keys. Note that when the Special Bit is cleared, the ASCII code in Data Word will correspond to upper or lower case ASCII codes independent of the state of these bits. That is, you do not need to check these bits to see if a character is upper case or not; the proper ASCII code is returned to you in the Data Word.
- RS:
- RP: This bit indicates that the device is auto-repeating and this key is a result of a previous key which has been held down. If auto-repeating keys are not acceptable to your application, you can check this bit and ignore the keypress if the bit was set.

DATA WORD. The format of the Data Word follows:

```

High Byte  Low Byte
+-----+
|  Null  |  Data  |
+-----+

```

The high order byte (Null) is null and will be all zeros.

The low order byte (Data) contains an ASCII code if the Special Bit was cleared or a Keycode value if the Special Bit was set.

If the Data is ASCII, the case will correspond to the proper upper or lower case ASCII code. There is no need to check the state of the Shift Bits.

Note that some languages ignore null input. BASIC is an example of such a language, and you will not be able to perform keycode input in these languages. It is suggested you use an assembler input routine to accept this data. You may further wish to use an input function which does not process Control-C. MS-DOS function 7 is an example of such a function. (See the *Microsoft Programmer's Manual* for details.)

Touchscreen Input in Keycode Mode

In Keycode mode, the touchscreen will return reports in the same format as when not in Keycode mode. However, each character will return four bytes (Qualifier Word and Data Word). Let's review each type of field and row/column reports.

ASCII FIELD. When Keycode mode is on, each character in the ASCII field buffer will be reported as four bytes. (This is another good reason to keep field lengths to a minimum!) The Device ID byte is set to OFF hex to indicate a touchscreen ASCII field.

KEYCODE FIELDS. This type of field allows you to specify a Qualifier Word and a Keycode Word. You will receive just those four bytes on return. The Device ID byte is set to 80 hex to indicate a touchscreen Keycode field.

TOGGLE AND NORMAL FIELDS. These two fields each return three-character reports in normal mode. The first data byte is the touch status (ON or OFF); the next two bytes are the defined response string. They are usually used to identify the touch field. In Keycode mode, you will receive four bytes for each of these three characters. In other words, you will receive 12 bytes of data for each report in Toggle fields and Normal fields. The Device ID byte is set to 80 hex to indicate touchscreen Toggle or Normal fields.

ROW/COLUMN REPORTS. In normal mode, you receive three-character row/column reports. The first data byte is the touch status (ON or OFF); the second byte is the touched row number; the third byte is the touched column number. When Keycode mode is on, you will receive four bytes for each of these three characters, which means you must process 12 characters for each report. The Device ID byte is set to 80 hex to indicate touchscreen row/column report.

Softkeys in Keycode Mode

The user-defined softkeys are reported as a buffer of text and are defined with the ESC & f escape sequence. You will find that, like ASCII touch fields, you receive four bytes for each character in the user-defined softkey buffer.

If you are using application softkeys, you will receive only a four-byte report, two keycode bytes and two qualifier bytes. Note that application softkeys will set the Special bit in the Qualifier word, and the Data word represents a keycode.

Turn Keycode Mode On/Off

This function turns Keycode mode of the console device ON or OFF.

Input Buffer:

(0,43)	Function Code
MODE	Keycode mode:
	1 = ON
	0 = OFF

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

If Keycode mode is OFF, each key pressed on the keyboard returns one byte of data when the console input device is read. If Keycode mode is ON, each keypress returns four bytes of data. The first two bytes form a word of qualifiers and the next two bytes form a word of key data. See Keycode Mode on the previous pages for more details.

Read Keycode Mode Status

This function returns the ON or OFF status of Keycode mode.

Input Buffer:

(0,44)

Function Code

((BUFFER))

A double-word pointer to a byte location where the Keycode mode ON/OFF status is returned.

First word = Buffer offset address

Second word = Buffer segment address

0 = Keycode mode OFF

1 = Keycode mode ON

Outputs:

AX = 0

Successful

AX <> 0

Unsuccessful

Read Numeric/Graphics Keypad Status

This function returns the status of whether the extened keypad is in Numeric mode or Graphics mode.

Input Buffer:

(0,45)	Function Code
((BUFFER))	A double-word pointer to a byte location where the keypad status is returned.
	First word = Buffer offset address
	Second word = Buffer segment address

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful
BUFFER	0 = numeric mode
	1 = graphics mode

Remarks:

Your application should set the desired setting of the Numeric/Graphics keypad. The keypad can be altered by the following escape sequences:

ESC & k 0 0	Turn on Numeric keypad
ESC & k 1 0	Turn on Graphics keypad

Read Key

This function returns the key qualifiers and the keycode of the key entered from the keyboard. Keycode mode can be ON or OFF.

Input Buffer:

(0,46)	Function Code
((BUFFER))	A double-word pointer to a 2-word buffer where the Qualifier word and the Keycode word will be returned.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful
(BUFFER)	
First Word	Qualifier
Second Word	Keycode

Remarks:

This function does not actually read (remove) characters from the keyboard input buffer; it just returns the key's information in a specified buffer. To flush the keyboard input buffer, your application has to perform console-reads.

This function is useful because it lets you examine the keycode and qualifiers without turning on Keycode mode.

NOTE

This function is included in Operating System Version B.01.00 or later.

GRAPHICS INPUT/OUTPUT SUBSYSTEM FUNCTION REFERENCE

SECTION

5

NELSI THT - EL

INTRODUCTION

This section lists functions in the Graphics Input and Output Subsystem (GIOS) in numerical order according to the function number. If you do not already know the procedure for calling Alphanumeric and Graphics Input and Output Subsystem (AGIOS) functions, and are not familiar with parentheses and positional notations used in the function reference, you should read Section 3 on AGIOS programming first.

GIOS FUNCTIONS

Each GIOS function is identified by a 16-bit function code as shown below:

```
Bit 15          0
+-----+
| 4 | X |
+-----+
high   low
byte   byte
```

For GIOS functions, the high byte of the function code is always a number "4". The low byte (X) is the function number. The function code must be the first parameter stored in the input buffer. The other parameters, if any, must be stored in the buffer in the order that is listed in the function description.

In this section, a GIOS function code is given in decimal and is represented by an ordered-pair (4,X). You should note that the Intel processor interprets the two bytes in a word in reverse order, that is, in memory, the value X is stored before the value 4.

There are two ways to define a function code; you can define it either in byte-mode or in word-mode using Microsoft Macro Assembler directives Define Byte (DB) or Define Word (DW).

GIOS Function Reference

For example, to allocate a buffer called `IN_BUF` and store the function code (4,1) in the first word of `IN_BUF`, you can use either one of the following instructions:

```
In Byte Mode:      IN_BUF  DB  1,4
```

```
In Word Mode:      IN_BUF  DW  0401H
```

The above two instructions both allocate a buffer called `IN_BUF` and store the value 1 in the first byte and the value 4 in the second byte of the buffer.

GIOS lets you programmatically control the following graphics functions:

- o Display Control
- o Vector Drawing Mode
- o Graphics Text
- o Graphics Plotting
- o Graphics Status

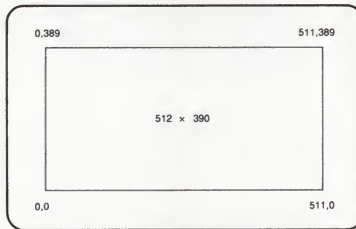
For information on using Graphics escape sequences, refer to Appendix D on Graphics Control Functions.

DISPLAY CONTROL (ESC * d)

The graphics and alphanumeric data are displayed in the same area on the screen, but they are stored in separate display memory. This lets you read or modify graphics and alphanumeric data separately.

The alphanumeric display memory has 48 lines; only 24 data lines are visible on the screen at any time. The softkey labels are located at rows 25 and 26 and the status line is located at row 27.

The graphics display memory size is 512 pixels (horizontal) by 390 pixels (vertical). The graphics display is actually larger in screen area than the alpha display.



GIOS Function Reference

This section describes the GIOS functions that support the alphanumeric and graphics displays. The following display control functions are explained in detail in this section.

<u>Function Code</u>	<u>Function</u>
(4,1)	Clear Graphics Memory
(4,2)	Set Graphics Memory
(4,3)	Turn On Graphics Display
(4,4)	Turn Off Graphics Display
(4,5)	Turn On Alphanumeric Display
(4,6)	Turn Off Alphanumeric Display
(4,7)	Turn On Graphics Cursor
(4,8)	Turn Off Graphics Cursor
(4,9)	Turn On Rubber Band Line
(4,10)	Turn Off Rubber Band Line
(4,11)	Move Graphics Cursor Absolute
(4,12)	Move Graphics Cursor Incremental
(4,13)	Turn On Alphanumeric Cursor
(4,14)	Turn Off Alphanumeric Cursor
(4,15)	Turn On Graphics Text Mode
(4,16)	Turn Off Graphics Text Mode

Since the alphanumeric and graphics displays and cursors operate independently, your application can use the above GIOS functions to set up the ON or OFF state of each one of them.

Clear Graphics Memory (ESC * d a)

This function clears graphics display memory. The entire displayable graphics area of 512 x 390 pixels is turned off.

Input Buffer:

(4,1)	Function Code
-------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function blanks out the graphics display memory.

Example:

The following assembly program segment builds an input buffer and calls function (4,1) to clear graphics memory.

```

;
;Build input buffer
;
IN_BUF    DW    0401H           ;function code
          .
          .
          .
;
;Set up registers to call GIOS. The input buffer's segment
;address is assumed to be in DS already.
;
          MOV    AX,4403H       ;I/O control write
          MOV    BX,1           ;device handle
          MOV    CX,2           ;2 bytes in input buffer
          MOV    DX,OFFSET IN_BUF ;input buffer offset address
          INT    21H            ;call MS-DOS to clear memory
          RET                   ;return

```

Set Graphics Memory (ESC * d b)

This function sets graphics display memory. The entire displayable graphics area of 512 x 390 pixels is turned on.

Input Buffer:

(4,2)	Function Code
-------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Setting graphics display memory turns the entire graphics display to green.

Turn On Graphics Display (ESC * d c)

This function turns on the graphics display. The data in graphics memory is not affected.

Input Buffer:

(4,3)	Function Code
-------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

If the graphics cursor is ON, then it will be displayed.

Alphanumeric data and cursor are not affected.

Turn Off Graphics Display (ESC * d d)

This function turns off the graphics display. The data in graphics memory is not affected.

Input Buffer:

(4,4)	Function Code
-------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

If the graphics cursor is ON, then it will be hidden.

Alphanumeric data and cursor are not affected.

Turn On Alphanumeric Display (ESC * d e)

This function turns on the alphanumeric display. The data in alphanumeric memory is not affected.

Input Buffer:

(4,5)	Function Code
-------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function also turns on the alphanumeric cursor.

Graphics data and cursor are not affected.

Turn Off Alphanumeric Display (ESC * d f)

This function turns off the alphanumeric display. The data in alphanumeric memory is not affected.

Input Buffer:

(4,6)	Function Code
-------	---------------

Outputs:

AX = 0	Successful
AX = <> 0	Unsuccessful

Remarks:

If the alphanumeric cursor is ON, then it will be hidden.

Graphics data and cursor are not affected.

Turn On Graphics Cursor (ESC * d k)

This function turns on the graphics cursor.

Input Buffer:

(4,7)	Function Code
-------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The graphics cursor is initially OFF (power on or hard reset). Turning the cursor on or off does not affect the data in graphics memory.

The graphics cursor may be toggled on and off by pressing the [Graph cursor] key on the Graphics/Numeric pad in Graphics Pad mode.

Alphanumeric data and cursor are not affected.

Turn Off Graphics Cursor (ESC * d l)

This function turns off the graphics cursor.

Input Buffer:

(4,8)	Function Code
-------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Alphanumeric data and cursor are not affected.

Turn On Rubber Band Line (ESC * d m)

This function turns on the rubber band line and graphics cursor.

Input Buffer:

(4,9)	Function Code
-------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The default state (power on or hard reset) for the rubber band line and graphics cursor is OFF.

The graphics rubber band line is a line with its fixed point at the current pen position and its moveable point at the graphics cursor. The "pen" is an imaginary drawing pen and the "current pen position" is the point that was last moved to or drawn to.

Turning on the graphics rubber band line turns on the graphics cursor and draws a rubber band line between the cursor and the current pen position. When either the cursor or the pen moves, the rubber band line moves also.

Turn Off Rubber Band Line (ESC * d n)

This function turns off the rubber band line.

Input Buffer:

(4,10)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The graphics cursor is not affected.

Move Graphics Cursor Absolute (ESC * d <x>,<y> o)

This function moves the graphics cursor to the specified absolute location.

Input Buffer:

(4,11)	Function Code
(X-COORD)	The X coordinate of the new cursor position expressed as an absolute number in the range from -16384 to 16383.
(Y-COORD)	The Y coordinate of the new cursor position expressed as an absolute number in the range from -16384 to 16383.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Although the value of X-COORD and Y-COORD parameters can be in the range of -16384 to +16383, the absolute display coordinates are from 0 to 511 in the X direction and from 0 to 389 in the Y direction. If you specify a value outside of these ranges, the cursor will be positioned at the edge of the screen in the absolute X or Y direction.

The cursor is moved even if it is turned off.

Move Graphics Cursor Incremental (ESC * d <x>,<y> p)

This function moves the graphics cursor to a specified location that is relative to the current cursor position.

Input Buffer:

(4,12)	Function Code
(X-COORD)	The X coordinate of the new cursor position expressed as a number that is relative to the current cursor position. Its range extends from -32768 to +32767.
(Y-COORD)	The Y coordinate of the new cursor position expressed as a number that is relative to the current cursor position. Its range extends from -32768 to +32767.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

In this function, X-COORD and Y-COORD are added to the current cursor position to produce a resultant position where the cursor is to be moved.

The cursor is moved even if it is turned off.

Turn On Alphanumeric Cursor (ESC * d q)

This function turns on the alphanumeric cursor.

Input Buffer:

(4,13)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The data in the alphanumeric memory is not affected.

Graphics data and cursor are not affected.

Turn Off Alphanumeric Cursor (ESC * d r)

This function turns off the alphanumeric cursor.

Input Buffer:

(4,14)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The data in the alphanumeric memory is not affected.

Graphics data and cursor are not affected.

Turn On Graphics Text Mode (ESC * d s)

This function turns on Graphics Text mode. Characters that normally go to the alphanumeric display will be drawn on the graphics display.

Input Buffer:

(4,15)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The text is drawn with the current text attributes. Text attributes can be defined using GIOS functions (4,29) - (4,33). Graphics Text mode is described later in the section on Graphics Text.

Turn Off Graphics Text Mode (ESC * d t)

This function turns off Graphics Text mode.

Input Buffer:

(4,16)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Graphics Text mode is described later in the section on Graphics Text.

VECTOR DRAWING MODE (ESC * m)

There are several parameters that can be set to allow a wide variety of drawing capabilities. These parameters define line or area patterns to be used when drawing vectors or polygons, position the relocatable origin, or define graphics text attributes.

This section describes the GIOS functions that support vector drawing. The following vector drawing functions are explained in detail in this section:

<u>Function Code</u>	<u>Function</u>
(4,17)	Select Drawing Mode
(4,18)	Select Line Type
(4,19)	Define Line Pattern and Scale
(4,20)	Define Area Fill Pattern
(4,21)	Fill Rectangular Area, Absolute
(4,22)	Fill Rectangular Area, Relocatable
(4,23)	Select Polygonal Fill Pattern
(4,24)	Select Boundary Pen
(4,25)	Set No Polygon Boundary
(4,26)	Set Relocatable Origin
(4,27)	Set Relocatable Origin to Current Pen Position
(4,28)	Set Relocatable Origin to Current Cursor Position

Appendix D has additional information on using Graphics Vector Drawing mode and (ESC * m) sequence.

Select Drawing Mode (ESC * m <mode> a)

This function selects the vector drawing mode.

Input Buffer:

(4,17)	Function Code
(MODE)	Drawing Mode:
	0 = Graphics memory not changed
	1 = Clear mode
	2 = Set mode
	3 = Complement mode
	4 = Jam mode

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Vectors can be drawn by setting, clearing, or complementing the data in the graphics memory. Normally the memory is cleared and vectors are drawn by setting selected bits to make green lines on a dark screen. If instead you want black vectors on a green screen, you can begin by setting graphics memory (Set Graphics Memory function (4,2)), select a clear mode, and draw dark vectors.

CLEAR MODE. Clear mode causes selected display bits to be turned off. The "selected bits" are those that are ON in the line pattern. (Function (4,18) lets you select line type.) If a solid line type (the default) has been selected, all of the bits in the vector will be selected. In clear mode this means that all of the dots making up a vector will be turned off. This allows you to draw dark vectors on a green background. Only those bits that are in the pattern are cleared. Bits that are OFF in the pattern do not affect the display.

SET MODE. Set mode is similar to clear mode except that the selected bits are turned on instead of off. Only the bits that are ON in the line type are affected.

COMPLEMENT MODE. Complement mode causes the selected display bits to change state (ON to OFF, OFF to ON). Again, only those bits that are ON in the line type or pattern are affected.

JAM MODE. Jam mode differs from the other modes in that both the bits that are ON and OFF in the line type or pattern affect the display. Jam mode has the effect of overlaying the display with the pattern.

NOTE

On the HP 150, Graphics Text drawn with Jam mode functions identically to Graphics Text drawn with Set mode. The illustration on the next page shows Jam mode used for Graphics Text.

SELECTIVE ERASE. A vector drawn in set mode can be selectively erased by redrawing it in clear mode. This will cause gaps to occur if the erased line is intersected by other lines. This problem can be overcome by initially drawing the line in complement mode and then redrawing it in complement mode to erase the line. This technique will preserve the original display. Complement mode is useful for drawing and erasing temporary figures.

***** See an illustration of drawing mode on the next page *****

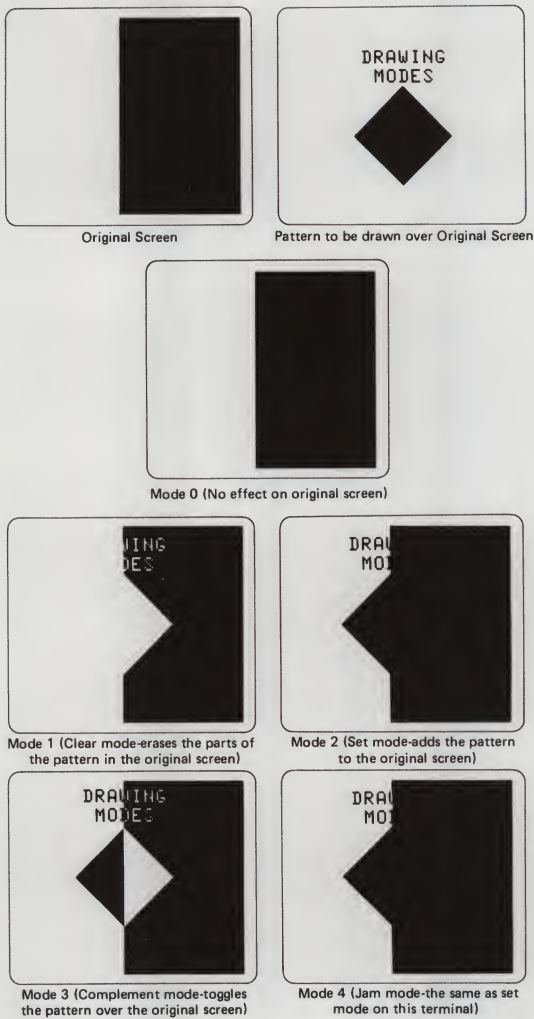


Figure 5-1. Graphics Drawing Mode

Select Line Type (ESC * m <type> b)

This function selects the vector line type.

Input Buffer:

(4,18)	Function Code
(TYPE)	Line Type
	1 Solid line (default)
	2 User defined line pattern
	3 Current area pattern
	4 Predefined pattern #1
	5 Predefined pattern #2
	6 Predefined pattern #3
	7 Predefined pattern #4
	8 Predefined pattern #5
	9 Predefined pattern #6
	10 Predefined pattern #7
	11 Point plot

1 =	_____
4 =	-----
5 =	-----
6 =	-----
7 =
8 =	-----
9 =
10 =	-----
11 =	(POINT PLOT)

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

You can select the dot pattern used when drawing vectors or filling rectangular areas. Dotted and dashed lines can be drawn by selecting one of nine predefined patterns or a user-defined line or area pattern. This allows you to use different line patterns to distinguish between groups of plotted data or to easily generate shading and cross hatching for use in engineering drawings, graphs or fabric patterns.

You can select one of eleven line types. The default line type is a solid line (type 1). Once the line type has been selected, all drawing vectors are drawn using that line type.

GIOS Function Reference

Point plot causes a single point to be plotted at the coordinates specified by the data. This line type is useful for generating "scattergram" graphs.

If current area shading is selected ('type = 3) the line patterns used are selected from the eight lines making up the area fill pattern (refer to Define Area Pattern function (4,20)). The display is divided into groups of eight rows and eight columns. Horizontal and vertical lines are drawn using the appropriate row or column from the area pattern. Diagonal lines are drawn using a solid vector. If a line is longer than 8 dots, the pattern is repeated to complete the vector. See the illustration below.

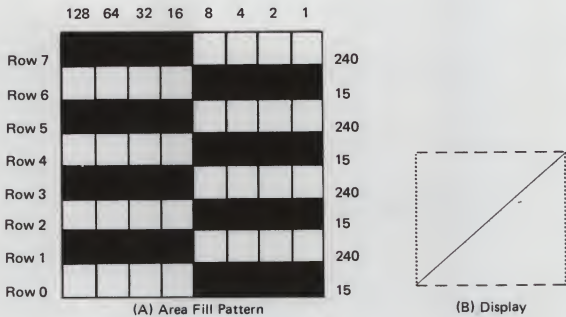


Figure 5-2. Area Fill Pattern

Define Line Pattern and Scale (ESC * m <pattern><scale> c)

This function defines a user line pattern and scale.

Input Buffer:

(4,19)	Function Code
(,PATTERN)	The line pattern is an integer in the range of 0 to 255. (The high byte is a "don't care".)
(SCALE)	The line scale is an integer in the range of 1 to 255.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The dot pattern used to draw vectors can be defined programmatically. Once a pattern is defined, you must select the user-defined line type (type = 2) using the Select Line Type function (4,18).

A user-defined line pattern is composed of a dot pattern and a scale factor. The dot pattern is a sequence of eight 1's and 0's. Using the default drawing mode (Set mode), points indicated as a "1" in the pattern are drawn, and the points indicated as a "0" are left unchanged.

The pattern is given as a decimal number between 0 and 255 that is the decimal equivalent of the 8-bit binary pattern. The default pattern is all "1"s (255). For example, = 10101010 (binary) = 170 (decimal). The actual number used for the pattern can be between -32768 and 32767. The least significant 8 bits of the number's 2's complement equivalent are used to determine the pattern.

The scale factor indicates how many times each bit in the pattern is repeated. For example, a scale factor of 3 applied to the pattern defined above results in a pattern of or 111000111000111000111000 (binary).

Example:

Define a pattern to generate the following vector:

```
11111111110011001111111111001100
pattern = 11111010 = 250
scale = 2
```

GIOS Function Reference

Those area patterns too complex to be obtained from an 8x8 area pattern can be generated by plotting a series of lines and varying the patterns used for successive lines. Complex patterns such as those used in weaving can be generated easily using this technique.



Figure 5-3. Examples of User-Defined Line Types

Define Area Fill Pattern (ESC * m <pattern> d)

This function defines a user area fill pattern of 8 dots by 8 dots (8x8) on the screen.

Input Buffer:

(4,20)	Function Code
(,DATA ROW0)	You specify all eight rows of the 8x8 fill pattern. Each DATA-ROW is an eight-bit byte which defines a particular row of the pattern. (The high byte is a "don't care".)
(,DATA ROW1)	
(,DATA ROW2)	
(,DATA ROW3)	
(,DATA ROW4)	
(,DATA ROW5)	
(,DATA ROW6)	
(,DATA ROW7)	

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The display is divided into 8 dot by 8 dot cells. Every point on the display is mapped to a corresponding bit in the pattern.

The user area pattern is defined by 8 parameters, one for every row of dots in the pattern. Each parameter is interpreted as a 2's complement number, and the least significant 8-bits are used to obtain a value between 0 and 255. The 8-bit number (0 to 255) represents an 8-bit binary pattern. Bits set to 1 are drawn, and bits set to 0 (zero) are not drawn (depending on the current drawing mode). Function (4,17) lets you select the current drawing mode.

The user area pattern can also be used to provide line patterns for horizontal or vertical lines when the area pattern is selected as a line type (type=3). (Refer to Select Line Type function (4,18) for additional information.) Drawing horizontal or vertical lines causes the corresponding row or column of the pattern to be used as the line pattern. Diagonal vectors will always be drawn using a solid line. Irregular shapes can also be built up by selecting the area shading pattern and then using successive horizontal and vertical lines.

The default area fill pattern is a solid-filled pattern.

***** See an example of this function on the next page *****

GIOS Function Reference

Example: Define a simple checkerboard pattern.

Row 0 = 10101010 = 170
Row 1 = 01010101 = 85
Row 2 = 10101010 = 170
Row 3 = 01010101 = 85
Row 4 = 10101010 = 170
Row 5 = 01010101 = 85
Row 6 = 10101010 = 170
Row 7 = 01010101 = 85

The 8080 assembly sample program segment to define the above pattern is as follows:

```
; Build the input buffer
IN_BUF DB 20,4                ;function code
        DW 170                ;DATA ROW0
        DW 85                 ;DATA ROW1
        DW 170                ;DATA ROW2
        DW 85                 ;DATA ROW3
        DW 170                ;DATA ROW4
        DW 85                 ;DATA ROW5
        DW 170                ;DATA ROW6
        DW 85                 ;DATA ROW7
        .
        .
        .
;
; Set up registers to call GIOS functions
; DS is assumed to have the default data segment
MOV AX,4403H                ;I/O Control write
MOV BX,1                    ;console handle
MOV CX,18                   ;buffer length
MOV DX,OFFSET IN_BUF        ;buffer address
INT 21H                     ;execute GIOS function
CMP 0                       ;check return status
JNZ ERROR                   ;process error
RET                          ;otherwise, return.
ERROR:
.
.
.
```

Other examples of user-defined patterns are shown below.

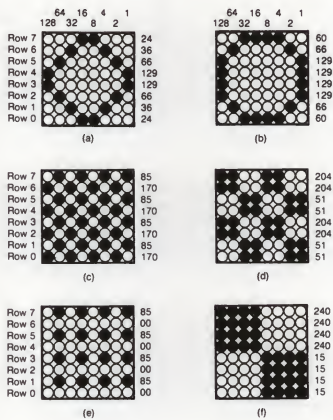


Figure 5-4. Examples of User-Defined Area Fill Patterns

Fill Rectangular Area, Absolute (ESC * m <x1>,<y1> <x2>,<y2> e)

This function fills a rectangular area with the selected area fill pattern. The rectangular region is defined by specifying the lower left and upper right coordinates.

Input Buffer:

(4,21)	Function Code
(LWR LEFT X-COORD) (LWR LEFT Y-COORD)	The X,Y absolute coordinates of the lower left corner of the rectangular area to be filled.
(UPR RIGHT X-COORD) (UPR RIGHT Y-COORD)	The X,Y absolute coordinates of the upper right corner of the rectangular area to be filled.
Each coordinate is in the range of -16384 to +16383.	

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

A rectangular area can be filled with one of a variety of predefined patterns or with a user-defined pattern. The pattern can also be used to provide line patterns for horizontal or vertical lines when the area pattern is selected as the line type. (Refer to Select Line Type function (4,18) and Define Area Fill Pattern function (4,20) for additional information.)

When an area fill pattern is selected, the entire screen is divided into 8x8 cells. Each location is mapped to the corresponding bit in the pattern. When an area fill operation is performed, the area fill pattern is duplicated to fill the area.

Fill Rectangular Area, Relocatable (ESC * m <x1>,<y1> <x2>,<y2> f)

This function fills a rectangular area with the selected area fill pattern. The rectangular region is defined by specifying the lower left and upper right coordinates.

Input Buffer:

(4,22)	Function Code
(LWR LEFT X-COORD)	The X,Y relocatable coordinates of the lower left corner of the rectangular area to be filled.
(LWR LEFT Y-COORD)	
(UPR RIGHT X-COORD)	The X,Y relocatable coordinates of the upper right corner of the rectangular area to be filled.
(UPR RIGHT Y-COORD)	

Each coordinate is in the range from -32768 to +32767.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The given coordinates are added to the relocatable origin. The resultant values are then treated as absolute coordinates. Functions (4,26), (4,27), (4,28), and (4,49) has additional information on relocatable origin.

A rectangular area can be filled with one of a variety of predefined patterns or with a user-defined pattern. The pattern can also be used to provide line patterns for horizontal or vertical lines when the area pattern is selected as the line type. (Refer to Select Line Type function (4,18) and Define Area Fill Pattern function (4,20) for additional information.)

When an area fill pattern is selected, the entire screen is divided into 8x8 cells. Each location is mapped to the corresponding bit in the pattern. When an area fill operation is performed, the area fill pattern is duplicated to fill the area.

Select Polygonal Fill Pattern (ESC * m <pattern> g)

This function selects a pattern for polygonal and rectangular area fill.

Input Buffer:

(4,23)	Function Code
(PATTERN)	Area Fill Pattern:
	1 = Solid fill pattern
	2 = User-defined fill pattern
	3 = Dotted hatching
	4 = Dashed hatching
	5 = Hatching
	6 = Cross hatching
	7 = Fine hatching
	8 = Medium checkerboard
	9 = Fine checkerboard, 1:1 blend
	10 = Fine checkerboard, 3:1 blend

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The default pattern for polygonal area fill is a user-defined pattern. Function (4,20) lets you define a user-defined area fill pattern.

Select Boundary Pen (ESC * m <pen> h)

This function selects the pen to be used to draw the boundary of a filled polygon. The boundary is drawn with a solid line pattern.

Input Buffer:

(4,24)	Function Code
(PEN)	Boundary Pen Number (see Remarks below).

Outputs:

AX = C	Successful
AX <> 0	Unsuccessful

Remarks:

Since the HP 150 is a black and green system, the selected pen number is not significant. Any number assigned to (PEN) will generate a solid green boundary line.

The default setting is no polygon boundary. Function (4,25) turns polygon boundary OFF.

Set No Polygon Boundary (ESC * m h)

This function turns off drawing of boundary around a polygon.

Input Buffer:

(4,25)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Function (4,24) lets you select (enable) the boundary pen.

Set Relocatable Origin (ESC * m <x>,<y> j)

This function sets the relocatable origin to the specified absolute location.

Input Buffer:

(4,26)	Function Code
(X-COORD)	The X coordinate is the new relocatable origin expressed as an absolute number in the range of -16384 to +16383.
(Y-COORD)	The Y coordinate is the new relocatable origin expressed as an absolute number in the range of -16384 to +16383.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The relocatable origin lets you use one set of data and drawing commands to display a figure at several different positions on the screen. The value of the relocatable origin is added to the relocatable data to obtain the coordinates used to draw the figure. Once you change the relocatable origin, you have to re-draw the figure in order to see the relocation.

The following figure illustrates the effect of a relocatable origin on the display:

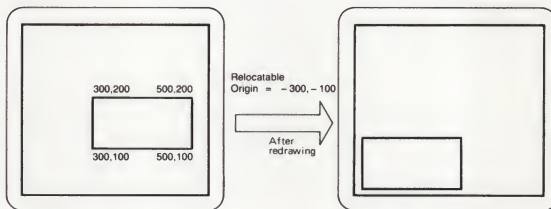


Figure 5-5. An Example of Using the Relocatable Origin

There are four GIOS functions for setting the graphics relocatable origin:

- o Set Relocatable Origin in Absolute Coordinates, function (4,26)
- o Set Relocatable Origin to Current Pen Position, functions (4,27) and (4,49)
- o Set Relocatable Origin to Current Cursor Position, function (4,28)

The default relocatable origin is at (0,0), the lower left corner of the display.

Set Relocatable Origin to Current Pen Position (ESC * m k)

This function sets the relocatable origin to the current pen position.

Input Buffer:

(4,27)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The current pen position is the last point moved to or drawn to.

The relocatable origin lets you use one set of data and drawing commands to display a figure at several different positions on the screen. The value of the relocatable origin is added to the relocatable data to obtain the coordinates used to draw the figure. Once you change the relocatable origin, you have to re-draw the figure in order to see the relocation.

Set Relocatable Origin to Current Cursor Position (ESC * m I)

This function sets the relocatable origin to the current graphics cursor position.

Input Buffer:

(4,28)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The relocatable origin lets you use one set of data and drawing commands to display a figure at several different positions on the screen. The value of the relocatable origin is added to the relocatable data to obtain the coordinates used to draw the figure. Once you change the relocatable origin, you have to re-draw the figure in order to see the relocation.

GRAPHICS TEXT (ESC * m)

The HP 150 features a full graphics character set implemented in Read-Only Memory (ROM). With graphics text, you can specify such attributes as slant, size, and orientation of characters sent to the graphics display. Graphics Text sequence (ESC * m) is described in Appendix D on Graphics Control Functions.

You can also create your own custom characters, either one at a time or as a replacement for the entire set stored in the HP 150.

This section gives the GIOS functions that support graphics text. The following graphics text functions are explained in detail in this section:

<u>Function Code</u>	<u>Function</u>
(4,29)	Set Graphics Text Size
(4,30)	Set Graphics Text Orientation
(4,31)	Turn On Text Slant
(4,32)	Turn Off Text Slant
(4,33)	Set Graphics Text Origin
(4,34)	Display Graphics Text Label
(4,35)	Define User Character Set
(4,36)	Select Default Character Set
(4,37)	Output Single Text Character
(4,38)	Set Graphics Defaults

Functions (4,15) and (4,16) listed earlier let you turn Graphics Text mode on and off, respectively.

Functions (4,72) and (4,73) listed later let you set picture definition defaults and perform Graphics hard reset, respectively.

Set Graphics Text Size (ESC * m <size> m)

This function sets the graphics text size.

Input Buffer:

(4,29)

Function Code

(X-SCALE)

The X coordinate (width) scale factor for text characters.

Bit	15	8	7	0

Bits 0-7 = fraction

Bits 8-15 = integer

(Y-SCALE)

The Y coordinate (height) scale factor for text characters.

Bits 0-7 = fraction

Bits 8-15 = integer

Outputs:

AX = 0

Successful

AX <> 0

Unsuccessful

Remarks:

The vector lists that define the user-defined character set are scaled using this text size. (See function (4,35) for more information on defining user character set.)

The "ESC * m <size> m" sequence only lets you set the text size to one of eight values that is prescaled by the system. This function lets you scale graphics text size to a precision that is not available via the escape sequence.

Example:

The following example calculates the X-SCALE and Y-SCALE parameters for a text size of 21 pixels by 28 pixels:

$$\begin{aligned} \text{X-SCALE} &= 3 \times 256 \\ &= 768 \\ &= 0300\text{H (in hex)} \end{aligned}$$

$$\begin{aligned} \text{Y-SCALE} &= 2.8 \times 256 \\ &= 717 \\ &= 02\text{CDH (in hex)} \end{aligned}$$

The system default character cell is 7 pixels wide by 10 pixels high. Therefore, the new text size of 21 pixels wide is exactly 3 times the default character cell width and 28 pixels is 2.8 times the default character cell height.

In order to move the integer portion of the scale to the high byte (bits 8-15), you can multiply the scale by 256.

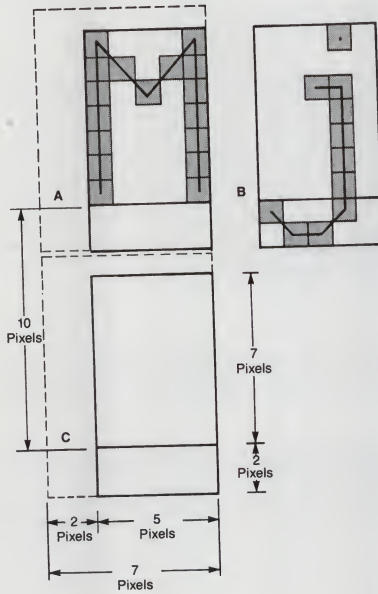


Figure 5-6. Graphics Text Size (Scale=1, Text Origin=0)

The figure above shows how a scale 1 graphics character is positioned on the screen.

- o Scale 1 characters are generally drawn in a 5 pixels by 7 pixels matrix (5x7) which is in a 7 pixels by 10 pixels (7x10) character cell.
- o The bottom 2 pixels are not used except for cases like the "j" character.
- o If the character (M) is drawn and the current pen position is at point A, the character will be placed in the cell as shown above. The current pen position is moved to point B.
- o If a carriage return and a line feed are issued, the current pen position will be moved to point C.
- o The characters are drawn with vectors, so that they can be scaled over a continuous range (i.e., 1.2, 1.3, ...). This also means the character strokes will not get proportionally thicker when scaled up. The width of a character stroke is the same as the width of a vector.

The above illustration shows how a scale 1 character is displayed. If you select scale 2, both the character and the cell size will be doubled. As you increase or decrease the text scale, the size of the character and the character cell will be scaled proportionally.

NOTE

A character cell whose scaled cell width is not an integer value (say 10.5 pixels wide) will be drawn on the display as a character with a cell width equal to the integer portion of the desired cell width (ie. 10 pixels wide). A string of these characters will, however, have its length adjusted by inserting extra pixels between appropriate adjacent characters. Therefore, a string of 30 characters whose scaled cell widths should be 10.5 pixels each will generate a string whose width is 315 pixels. Similar adjustments occur with the character cell height.

Set Graphics Text Orientation (ESC * m <orientation> n)

This function selects the graphics text orientation. This also changes the direction of line feed, carriage return, and backspace.

Input Buffer:

(4,30)	Function Code
(ORIENTATION)	Graphics Text Orientation:
	1 = Normal (horizontal)
	2 = Rotate 90 degrees counterclockwise
	3 = Rotate 180 degrees counterclockwise
	4 = Rotate 270 degrees counterclockwise

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The following diagram shows the four different orientations.

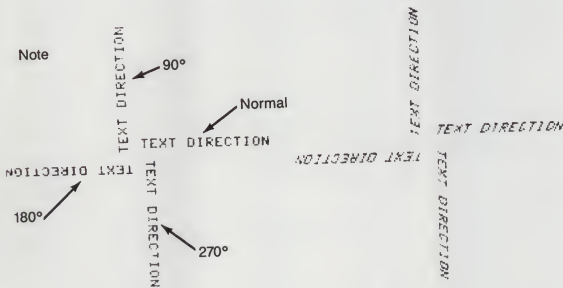


Figure 5-7. Graphics Text Orientation

Turn On Text Slant (ESC * m o)

This function turns on the 26.57 degree slant of graphics text characters.

Input Buffer:

(4,31)

Function Code

Outputs:

AX = 0

Successful

AX <> 0

Unsuccessful

Turn Off Text Slant (ESC * m p)

This function turns off 26.57 degrees slant of graphics text characters.

Input Buffer:

(4,32)

Function Code

Outputs:

AX = 0

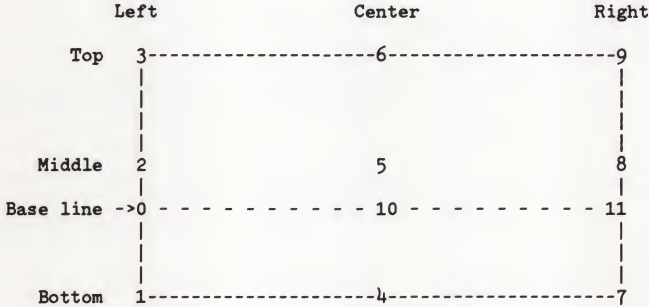
Successful

AX <> 0

Unsuccessful

Set Graphics Text Origin (ESC * m <origin> q)

This function sets the graphics text origin to one of twelve positions of text justification. The positions are shown in this figure:



Input Buffer:

(4,33)	Function Code
(ORIGIN)	Graphics Text Origin:
	A number from 0 to 11.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Text strings can be automatically right or left justified, or centered about a specified point. The value given in the ORIGIN parameter indicates the justification and placement of characters with respect to the current pen position.

If text is left justified, the current pen position is the left margin. Center causes the text to be centered on the pen position. Right justify selects the pen position as the right margin. Bottom, base line, middle and top select the part of the text string to be aligned with the current pen position.

Display Graphics Text Label (ESC * I <text>)

This function outputs a string of graphics characters. The label is drawn beginning at the current pen position.

Input Buffer:

(4,34)	Function Code
((TEXT))	Segment and offset address of a string of characters. The string must be terminated by Carriage Return (CR), Line Feed (LF), CR LF, or LF CR. The maximum length for TEXT is 73.
	First Word = Offset
	Second Word = Segment

Outputs:

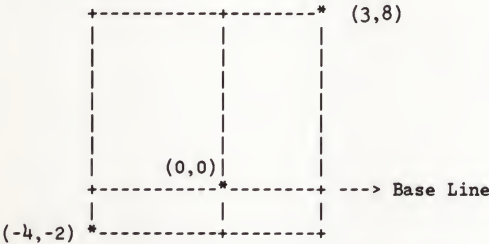
AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The text label is drawn with the current text attributes. Text attributes can be set up using GIOS functions (4,29), (4,30), (4,31), (4,32), (4,33).

Creating a Graphics Character

In order to create your own graphics character set, let's first look at the format of a single graphics text cell. A graphics character cell is 7 pixels wide by 10 pixels high.



Each graphics character is stored as a vector list describing how the character is formed. Within this list, you must describe the character within a cell. The vector list format is:

X L L
Y L L
X U R
Y U R
X 1
Y 1
X 2
Y 2
X n
Y n

XLL, YLL, XUR, and YUR are coordinates for the lower left and upper right points of the character cell used in placing the character within the character grid.

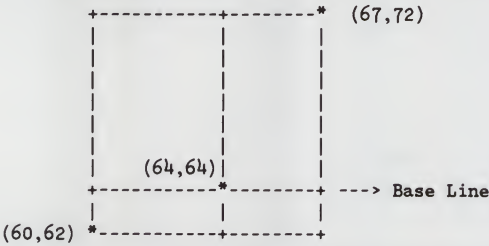
(X1,Y1), (X2,Y2) ..., (Xn,Yn) are ordered pairs that describe the character strokes.

The last (X,Y) must be (0,0) to indicate the end of the character stroke.

Each value in the above vector list is stored in "excess 64's notation", which means you must add 64 to the relative coordinate within each grid.

The Vector List

In the vector list illustrated above, the first four entries describe the cell size of a character of scale 1. These four bytes specify the lower left and the upper right coordinates. We can create a character cell in excess 64's notation. This character cell is:



Given this 'standard size cell', the first four entries in a vector list describing a character would be:

60	-> Lower left X coordinate
62	-> Lower left Y coordinate
67	-> Upper right X coordinate
72	-> Upper right Y coordinate

After these first four bytes, the vector list specifies an (X,Y) coordinate pair for each endpoint in the character. The X coordinate represents the X value and the Y coordinate represents the Y value, both in excess 64's notation.

The bit structure for each coordinate pair is:

Bit	7	6	0
	+-----+		
	M/D	X-COORDINATE	
	+-----+		
	0	Y-COORDINATE	
	+-----+		

M/D	1 = Move, 0 = Draw
X-COORDINATE	X coordinate in excess 64 representation
Y-COORDINATE	Y coordinate in excess 64 representation

If the most significant bit (bit 7) of the X coordinate is a "0", then the (X,Y) pair specifies a Draw mode. The 'pen' draws to the (X,Y) point. If the most

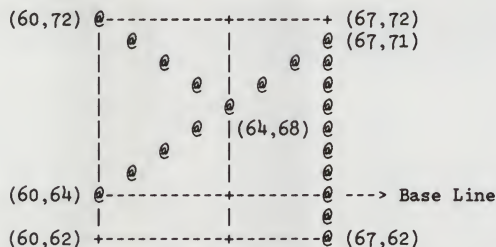
significant bit (bit 7) is a "1", then the (X,Y) pair indicates a Move mode. The pen is lifted and moved to (X,Y).

NOTE

The (X,Y) pair used to draw a character does not have to be in the range specified by the first four values (XLL, YLL, XUR, YUR) because these four values are only used to place the character.

Building a Character

The following example builds a single character using the character grid as shown earlier:



The steps in building a character vector list are listed below:

- o Start at the left base line by moving to (60,64)
- o Draw to (67,71)
- o Draw to (67,62)
- o Move the pen to (60,72)
- o Draw to (64,68)

The following program segment defines the vector list for the character illustrated above.

```

CHAR    DB      60                ;X Lower Left
        DB      62                ;Y Lower left
        DB      67                ;X Upper Right
        DB      72                ;Y Upper Right
;
; Note the + 128 indicates setting the 'move' bit
; + 0 indicates clearing the 'move' bit (draw)
;
        DB      60 + 128          ;Move to (60,64)
        DB      64                ;
        DB      67 + 0            ;Draw to (67,71)
        DB      71                ;
        DB      67 + 0            ;Draw to (67,62)
        DB      62                ;
        DB      60 + 128          ;Move to (60,72)
        DB      72                ;
        DB      64 + 0            ;Draw to (64,68)
        DB      68                ;
        DB      0                 ;Last X
        DB      0                 ;Last Y

```

Once you have built this list, you can use GIOS function (4,37) to output a single character. The character will be sent to the screen using the orientation, size, and slant characteristics that apply to graphics text.

Re-defining the Entire Character Set

The HP 150 character set is defined by a table of 256 entries. The contents of this table are 16-bit pointers to character vector lists as described earlier. In fact, the entry into the Pointer Table is the ASCII code of the character. For example, the 65th entry in the table of pointers corresponds to ASCII code 65 decimal, which is the character 'A'.

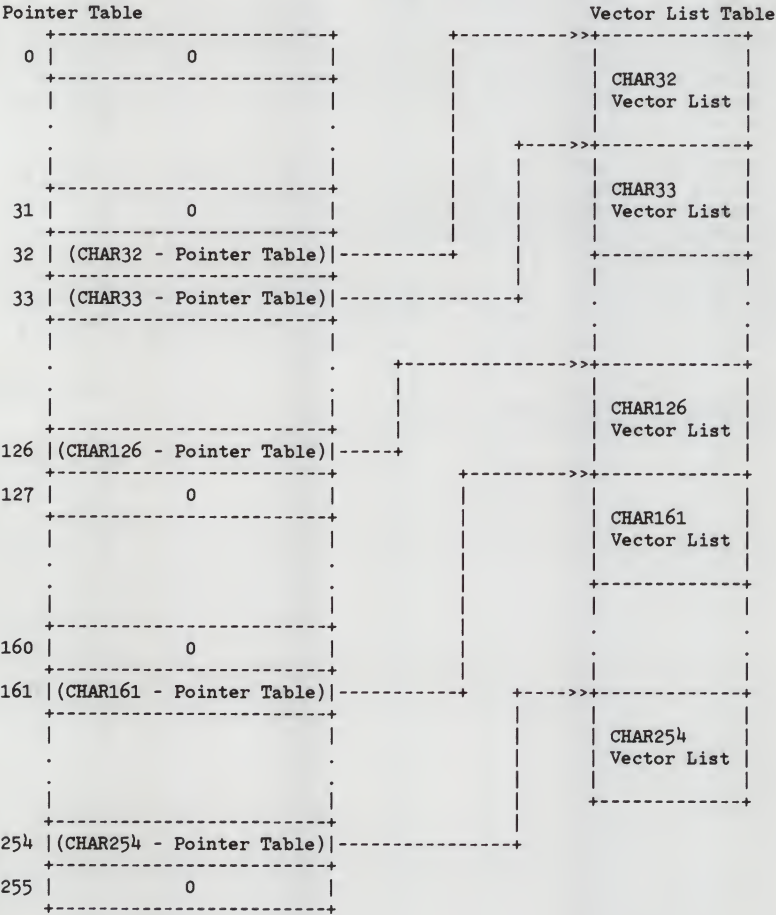
To re-define the HP 150 character set, you need to build a table of pointers to take the place of the ROM-based table. This entails creating your own table of 256 16-bit pointers. A pointer to a character vector list contains the difference between the address of that character vector list and the starting address of the Pointer Table.

For example, if the character 'A' vector list is defined at location 1750 and the Pointer Table's starting address is set up at location 500, then the 65th entry of the Pointer Table contains the value 1250.

Character 'A' Vector List	:	1750
Pointer Table Starting Address:	:	500
65th Table Entry Content	:	1750-500 = 1250

GIOS Function Reference

The following figure shows the relationship between the Pointer Table and the Vector List Table:



An entry in the Pointer Table contains either 0 or the pointer to the vector list that corresponds to its 8-bit ASCII code. A table entry that contains zero is a 'null pointer' and indicates that no vector list is defined for this table entry.

The first 32 entries in the table of pointers correspond to the ASCII control characters of values 0 through 31. These characters are not part of the HP 150 graphics character set, but the table entries must exist and contain zero. The same is true for ASCII values 128 through 159. Pointers 160 through 255 correspond to HP Roman-8 extensions. (You can usually access the HP Roman-8 extensions using the [Extend char] key on the keyboard.) If your application does not use 8-bit ASCII codes, pointers 160 through 255 should be set to null.

The 32nd character, ASCII Space, is an important entry in the table. The system uses its (XLL,YLL) and (XUR,YUR) values to calculate the new pen position when a back space or a line feed control character is received. The Space character will be displayed in place of any of the ASCII codes (33-126 and 161-254) whose pointers are null in the Pointer Table. Because of these special functions, a vector list must be defined for the Space character when defining a graphics character set. If the Space character is not defined, the system will return an error status when you use GIOS function (4,35) to define a graphics character set.

The system default Space character's vector list pointed by the 32nd entry of the system's Pointer Table is defined as follows:

CHAR32	DB	60	;XLL Coordinate
	DB	62	;YLL Coordinate
	DB	67	;XUR Coordinate
	DB	72	;YUR Coordinate
	DB	0	;Last X Byte
	DB	0	;Last Y Byte

The other non-zero entries in the Pointer Table contain the relative offsets (16 bit) of the vector lists that re-define the characters. You should remember that the offset is *relative to the start of the Pointer Table*, not an absolute offset.

Once you have defined such a table, you can use GIOS call (4,35) to re-define the graphics character set. In that function, you specify the offset and segment addresses of the first entry of the Pointer Table. (Remember: this first entry represents ASCII code 0. You must have 32 null pointers at the top of your list, and again in entries 128 through 159.) The Pointer Table and the Vector List Table must reside in the same 64K block of memory.

Programming Considerations

When creating graphics characters, you do not have to set the values of (XLL,YLL) and (XUR,YUR) to the corresponding values of the 'Space' character cell. For example, to implement proportional spacing within graphics text, you might define some characters to be wider and others narrower than the standard cell size.

In fact, the values of (XLL,YLL) and (XUR,YUR) are used only for placing characters next to each other on the display. The base line of all of the characters in a text string will be aligned.

Finally, if you replace the entire character set, remember that the offsets stored in the Pointer Table must specify an offset *relative to the start of the Pointer Table, not an absolute offset!*

NOTE

The numbers (coordinates) used in this section are basically what the HP 150 uses to generate graphics characters. However, if you want to build characters precisely the way you want them to appear on the screen, you may have to experiment with the data in the character vector lists.

Define User Character Set

This function lets you re-define the entire graphics character set. All subsequent graphics text operations will use this character set. This includes text size, orientation, slant, and justification.

Input Buffer:

(4,35)	Function Code
((TABLE))	Segment and offset address of the table that points to the Vector Lists of characters.
	First Word = Offset
	Second Word = Segment

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The previous pages contain information on defining a user character set with a Pointer Table and a Vector List Table.

Select Default Character Set

This function sets the character set to the default set maintained by the system. The cell size is 7 x 10.

Input Buffer:

(4,36)

Function Code

Outputs:

AX = 0

Successful

AX <> 0

Unsuccessful

Output Single Text Character

This function outputs a single graphics character defined by a vector list. All current graphics text operations such as size and orientation apply.

Input Buffer:

(4,37)	Function Code
((CHARACTER))	Segment and offset address of the vector list of a single character.
	First Word = Offset
	Second Word = Segment

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The previous pages contain information on creating and outputting graphics characters.

Example:

The following 8088 assembly program segment outputs the single character defined earlier in the section on Building a Character.

```

;
;Build the input buffer.
;
IN_BUF  DW  0437H           ;store function code first
        DW  CHAR           ;offset address of the character vector
        DW  0              ;Assume segment address=0
;
;Set up registers to call GIOS.
;Assume buffer's segment address is already in DS.
;
        MOV AX,4403H       ;I/O control write
        MOV BX,1           ;Console handle
        MOV CX,6           ;# of bytes in buffer
        MOV DX,OFFSET IN_BUF ;input buffer address
        INT 21H            ;output single character
        CMP AX,0           ;check for return status
        JNZ ERROR         ;process error
        RET               ;successful return
ERROR:  .
        .
        .

```

Set Graphics Defaults (ESC * m r)

This function sets the graphics parameters to their default values.

Input Buffer:

(4,38) Function Code

This function affects the following graphics parameters:

PARAMETER	DEFAULT VALUE
xx-Pen Condition	DOWN
xx-Line Type	1 (solid)
xx-Drawing Mode	2 (SET)
xx-User Line Pattern	255,1
xx-Area Fill Type	2 (user-defined pattern)
xx-User Area Fill Pattern	255,255....(solid)
xx-Boundary Pen	OFF
xx-Text Size	1
xx-Text Orientation	1
xx-Text Origin	1 (left bottom)
xx-Text Slant	0 (OFF)
xx-Graphics Text	OFF
Relocatable Origin	0,0
Graphics Display	ON
Graphics Cursor	OFF
Graphics Cursor Address	0,0
Rubberband Line	OFF

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Function (4,72) sets just those xx- parameters to their default values.

You should initialize the alphanumeric display and cursor to the state you want (ON/OFF) with GIOS function (4,5) or (4,6) and function (4,7) or (4,8).

GRAPHICS PLOTTING (ESC * p)

Graphic data is made up of vectors (line segments). The HP 150 uses the concept of a "pen" in drawing vector data. When enough parameter bytes (usually X,Y coordinates) have been received to specify a data point, the pen is moved from its current position to the new end point. If the pen is down, a vector will be drawn. If the pen is up, the pen is moved to the new point (without drawing a vector) and lowered. In either case, the new point becomes the current pen position.

Current Pen Position

Current pen position is specified in one of the following formats:

ABSOLUTE: The value in the ASCII absolute format can range from -16384 to 16383. Note that only points where X is in the range from 0 to 511 and Y is in the range from 0 to 389 will be visible on the screen.

INCREMENTAL: The values in the incremental format are added to the current pen position to obtain a new end point.

RELOCATABLE: The relocatable format allows you to use the relocatable origin to be added to the X and Y coordinates. The resultant values are then treated as absolute coordinates by the terminal. The relocatable format lets you use one set of data and drawing commands to display a figure at several different positions on the screen by changing the relocatable origin.

GIOS Function Reference

This section gives the GIOS functions that support graphics plotting. The following graphics plotting functions are explained in detail in this section:

<u>Function Code</u>	<u>Function</u>
(4,39)	Lift Pen
(4,40)	Vector Move, Absolute
(4,41)	Vector Move, Incremental
(4,42)	Vector Move, Relocatable
(4,43)	Lower Pen
(4,44)	Vector Draw, Absolute
(4,45)	Vector Draw, Incremental
(4,46)	Vector Draw, Relocatable
(4,47)	Plot to Cursor Position
(4,48)	Point Plot
(4,49)	Set Relocatable Origin to Current Pen Position
(4,50)	Start Polygonal Area Fill
(4,51)	Terminate Polygonal Area Fill
(4,52)	Polygon Move, Absolute
(4,53)	Polygon Move, Incremental
(4,54)	Polygon Move, Relocatable
(4,55)	Polygon Draw, Absolute
(4,56)	Polygon Draw, Incremental
(4,57)	Polygon Draw, Relocatable
(4,58)	Lift Boundary Pen
(4,59)	Lower Boundary Pen

Graphics plotting (ESC * p) sequence is described in Appendix D on Graphics Control Functions.

Lift Pen (ESC * p a)

This function lifts the pen.

Input Buffer:

(4,39)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The pen is an imaginary plotting pen. Movement of the pen from the current position will not draw a line.

Functions (4,40), (4,41), (4,42) let you lift the pen and move it to a specified location.

Vector Move, Absolute (ESC * p a <x>,<y>)

This function lifts the pen from the current pen position and moves the pen to an absolute coordinate position. The pen is lowered at the end of the operation.

Input Buffer:

(4,40)	Function Code
(X-COORD)	The X and Y numbers give an absolute coordinate position in the range from -16384 to +16383.
(Y-COORD)	

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Although the X-COORD and Y-COORD parameters can be in the range from -16384 to +16383, only points where X-COORD is in the range from 0 to 511 and Y-COORD is in the range from 0 to 389 will be visible on the screen.

Vector Move, Incremental (ESC * p a <x>,<y>)

This function lifts the pen from the current pen position and moves the pen to an incremental coordinate position. The pen is lowered at the end of the operation.

Input Buffer:

(4,41)

Function Code

(X-COORD)

(Y-COORD)

The X and Y numbers give an incremental coordinate position in the range from -32768 to +32767.

Outputs:

AX = 0

AX <> 0

Successful

Unsuccessful

Remarks:

This function adds X-COORD and Y-COORD to the current pen position to obtain a new pen position.

Vector Move, Relocatable (ESC * p a <x,y>)

This function lifts the pen and moves the pen to a relocatable coordinate position. The pen is lowered at the end of the operation.

Input Buffer:

(4,42)

Function Code

(X-COORD)

The X and Y numbers are a relocatable coordinate position in the range from -32768 to +32767.

(Y-COORD)

Remarks:

This function adds the relocatable origin to the incoming X-COORD and Y-COORD values. The resultant values are then treated as absolute coordinates.

If a relocatable origin has not been defined, the default relocatable origin (0,0) is used. You can use function (4,26), (4,27), (4,28), or (4,49) to define a relocatable origin.

Lower Pen (ESC * p b)

This function lowers the pen.

Input Buffer:

(4,43)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Functions (4,44), (4,45), (4,46) let you lower the pen and draw a vector to a specified location.

Vector Draw, Absolute (ESC * p b <x>,<y>)

This function lowers the pen and draws a vector to the coordinate position. The pen is lowered at the end of the operation.

Input Buffer:

(4,44)	Function Code
(X-COORD)	The X and Y numbers give the absolute coordinates of the
(Y-COORD)	vector position. They are in the range from -16384 to
	+16383.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

Note that only points where X-COORD is in the range 0 to 511 and Y-COORD is in the range from 0 to 389 will be visible on the screen.

Vector Draw, Incremental (ESC * p b <x>,<y>)

This function lowers the pen and draws a vector to the incremental coordinate position. The pen is lowered at the end of the operation.

Input Buffer:

(4,45)

Function Code

(X-COORD)

(Y-COORD)

The X and Y numbers give the incremental coordinates of the vector position. They are in the range from -32768 to +32767.

Outputs:

AX = 0

AX <> 0

Successful

Unsuccessful

Remarks:

This function adds X-COORD and Y-COORD to the current pen position to obtain a new pen position.

Vector Draw, Relocatable (ESC * p b <x>,<y>)

This function lowers the pen and draws a vector to the relocatable coordinate position. The pen is lowered at the end of the operation.

Input Buffer:

(4,46)	Function Code
(X-COORD)	The X and Y numbers give the relocatable coordinates of
(Y-COORD)	the vector position. They are in the range from -32768
	to +32767.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function adds the relocatable origin to the incoming X-COORD and Y-COORD values. The resultant values are then treated as absolute coordinates.

If a relocatable origin has not been defined, the default relocatable origin (0,0) is used. You can use function (4,26), (4,27), (4,28), or (4,49) to define a relocatable origin.

Plot to Cursor Position (ESC * p c)

This function moves the pen from its current position to the current cursor position if the pen is up. A draw is performed from the current pen position to the current cursor position if the pen is down.

Input Buffer:

(4,47)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Point Plot (ESC * p d)

This function draws a dot at the current pen position and then lifts the pen.

Input Buffer:

(4,48)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Set Relocatable Origin to Current Pen Position (ESC * p e)

This function sets the relocatable origin to the current pen position.

Input Buffer:

(4,49)

Function Code

Outputs:

AX = 0

Successful

AX <> 0

Unsuccessful

Start Polygonal Area Fill (ESC * p s)

This function starts polygonal area fill. The boundary pen is lowered with this function.

Input Buffer:

(4,50)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

You can define a polygon with as many as 105 sides. This function causes subsequent coordinate pairs to be read as vertices of the polygon. When a lift pen command (function (4,39)) occurs in the middle of a polygon area fill function, a new polygon is started. The Terminate Area Fill command (4,51) causes the polygon to be filled using the current drawing mode and area pattern.



Figure 5-8. Polygon Area Fill Example

If the polygon definition crosses over itself, the areas are defined in alternate order.



Figure 5-9. Overlapping Polygon Area Fills

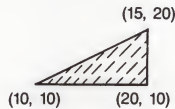
The only graphics functions allowed between Start Polygon Area Fill function (4,50) and Terminate Polygon Area Fill function (4,51) are:

Lift Pen: (4,39)
 Lower Pen: (4,43)
 Plot to Cursor Position: (4,47)
 Set Relocatable Origin to Current Pen Position: (4,49)
 Polygon Move functions: (4,52), (4,53), (4,54)
 Polygon Draw functions: (4,55), (4,56), (4,57)
 Lift Boundary Pen function: (4,58)
 Lower Boundary Pen function: (4,59)

Any other functions will terminate the polygon area fill function.

Example:

This example lists the procedural steps to draw a triangle at locations (10,10), (20,10), and (15,20) with area fill pattern 5, and with boundary pen enabled:



The procedural steps to execute the polygonal area fill is listed below:

1. Select polygonal fill pattern 5 using function (4,23)
2. Start polygonal area fill using function (4,50)
3. Move to point (10,10) using function (4,52)
4. Draw to point (20,10) using function (4,55)
5. Draw to point (15,20) using function (4,55)
6. Terminate polygonal area fill using function (4,51)

Note that it is not necessary to draw to point (10,10) after step 5 because the Terminate Polygonal Area Fill function automatically draws from the last point (15,20) to the first point (10,10).

Terminate Polygonal Area Fill (ESC * p t)

This function closes the current polygon definition and fills the polygon. The boundary pen is lifted at the end of this function.

Input Buffer:

(4,51)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The polygon is not drawn and filled until this function is called.

It is not necessary to specify a vector from the last point back to the first point; this function automatically closes the polygon definition. (See the example in the Start Polygonal Area Fill function (4,50) for more details.)

Polygon Move, Absolute

This function closes the polygon defined up to this point and moves the pen to the given absolute position to start a new polygon. If the pen is up, it is lowered at the end of this function.

Input Buffer:

(4,52)

Function Code

(X-COORD)

(Y-COORD)

The X and Y numbers give the absolute coordinates of the new position. They are in the range from -16384 to +16383.

Outputs:

AX = 0

AX <> 0

Successful

Unsuccessful

Remarks:

Although the X-COORD and Y-COORD parameters can be in the range from -16384 to +16383, only points in the absolute display region will be visible on the screen. That is, X-COORD is in the range from 0 to 511 and Y-COORD is in the range from 0 to 389.

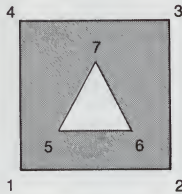
Polygon move is used at the start of a polygon and can be used in the middle to draw more complex polygons with "islands".

***** See an example of this function on the next page *****

GIOS Function Reference

Example:

The following example lists the procedural steps to draw a triangle inside a rectangle using polygon move functions (4,52), (4,53), or (4,54), and polygon draw functions (4,55), (4,56), or (4,57). The area outside of the triangle is filled with the currently defined area-fill pattern. The boundary pen is assumed to be enabled.



The steps to generate the above figure is listed below:

1. Start polygonal area fill using (4,50)
2. Move to point 1 using (4,52)
3. Draw to point 2 using (4,56)
4. Draw to point 3 using (4,56)
5. Draw to point 4 using (4,56)
6. Move to point 5 using (4,52)
7. Draw to point 6 using (4,56)
8. Draw to point 7 using (4,56)
9. Terminate polygonal area fill using (4,51)

Note that it is not necessary to draw to point 1 from point 4 after step 5. Move polygon function automatically closes the rectangular polygon.

Polygon Move, Incremental

This function closes the polygon defined up to this point and moves the pen to the given incremental coordinate position to start a new polygon. If the pen is up, it is lowered at the end of this function.

Input Buffer:

(4,53)	Function Code
(X-COORD)	The X and Y numbers give the incremental coordinates of
(Y-COORD)	the new position. They are in the range from -32768 to +32767.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function adds X-COORD and Y-COORD to the current pen position to obtain a new pen position.

Polygonal Move, Relocatable

This function closes the polygon defined up to this point and moves the pen to the given relocatable coordinate position to start a new polygon. If the pen is up, it is lowered at the end of this function.

Input Buffer:

(4,54)	Function Code
(X-COORD)	The X and Y numbers give the relocatable coordinates of
(Y-COORD)	the new position. They are in the range from -32768 to
	+32767.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function adds the relocatable origin to the incoming X-COORD and Y-COORD values. The resultant values are then treated as absolute coordinates.

If a relocatable origin has not been defined, this function uses the default relocatable origin (0,0). You can use function (4,26), (4,27), or (4,28) to define a relocatable origin.

Polygon Draw, Absolute

This function defines the edge of a polygon from the current pen position to the given absolute coordinate position.

Input Buffer:

(4,55)

Function Code

(X-COORD)

(Y-COORD)

The X and Y numbers give the absolute coordinates of the new position. They are in the range from -16384 to +16383.

Outputs:

AX = 0

AX <> 0

Successful

Unsuccessful

Remarks:

Note that only points where X-COORD is in the range from 0 to 511 and Y is in the range from 0 to 389 will be visible on the screen.

You can use polygon move and draw functions to draw more complex polygons. Remember, the polygon is not drawn until the Terminate Polygonal Area Fill function (4,51) is called. (See an example of a polygon with an "island" in function (4,52)).

Polygon Draw, Incremental

This function defines the edge of a polygon from the current pen position to the given incremental coordinate position.

Input Buffer:

(4,56)	Function Code
(X-COORD)	The X and Y numbers give the incremental coordinates of
(Y-COORD)	the new position. They are in the range from -32768 to
	+32767.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function adds X-COORD and Y-COORD to the current pen position to obtain a new pen position.

Remember, the polygon is not drawn until the Terminate Polygonal Area Fill function (4,51) is called.

Polygon Draw, Relocatable

This function defines the edge of a polygon from the current pen position to the given relocatable coordinate position.

Input Buffer:

(4,57)	Function Code
(X-COORD)	The X and Y numbers give the relocatable coordinates of
(Y-COORD)	the new position. They are in the range from -32768 to +32767.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function adds the relocatable origin to the incoming X-COORD and Y-COORD values. The resultant values are then treated as absolute coordinates.

Remember, the polygon is not drawn until the Terminate Polygonal Area Fill function (4,51) is called.

Lift Boundary Pen (ESC * p u)

This function lifts the polygon boundary pen. Edges of the polygon are not drawn. This remains in effect until the boundary pen is lowered.

Input Buffer:

(4,58)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

GIOS function (4,59) lets you lower the boundary pen.

Example:

The procedural steps to generate the following figure is listed below:



1. Enable polygon boundary pen, using (4,24).
2. Start polygonal area fill, using (4,50).
3. Move to point A, using (4,52).
4. Draw to point B, using (4,56).
5. Lift boundary pen, using (4,58).
6. Draw to point C, using (4,56).
7. Draw to point D, using (4,56).
8. Lower boundary pen, using (4,59).
9. Lift pen, using (4,39).
10. Move to point E, using (4,52).
11. Draw to point F, using (4,56).
12. Draw to point G, using (4,56).
13. Lift boundary pen, using (4,59).
14. Terminate polygonal area fill, using (4,51).

Step 10 moves the pen to point E and then lowers the pen; therefore, you do not have to specifically lower the pen.

The following assembly program implements the above procedure:

```

;
;This example uses GIOS functions to draw a triangle inside a rectangle.
;The GIOS functions used are Start Polygonal Area Fill, Lift and Lower
;Pen, Lift and Lower Boundary Pen, Polygonal Move and draw, and
;Terminate Polygonal Area Fill.
;
DATA          SEGMENT                ;start data segment
;
;Build input buffers for the series of AGIOS functions.
;
CMD_BUF       DB      3,4            ;function code - graphics display on
;
;Enable the boundary pen.
;
                DB      24,4          ;function code - enable boundary pen
                DW      0              ;pen number
;
;Select area fill pattern.
;
                DB      23,4          ;function code - select fill pattern
                DW      4              ;select pattern 4
;
;Start polygonal area fill.
;
                DB      50,4          ;function code - start polygon area fill
;
;Move to point A (100,100) of the rectangle.
;
                DB      52,4          ;function code - absolute code
                DW      100            ;X coordinate
                DW      100            ;Y coordinate
;
;Draw to point B (200,100) of the rectangle.
;
                DB      56,4          ;function code - incremental draw
                DW      100            ;X coordinate (right 100 pixels)
                DW      0              ;Y coordinate (no change)
;
;Lift up the boundary pen.
;
                DB      58,4          ;function code - lift boundary pen
;
;Draw to point C (200,250) of the rectangle.
;
                DB      56,4          ;function code - incremental draw
                DW      0              ;X coordinate (no change)
                DW      150           ;Y coordinate (up 150 pixels)
;
;Draw to point D (100,250) of the rectangle.
;
                DB      56,4          ;function code - incremental draw
                DW      -100           ;X coordinate (left 100 pixels)
                DW      0              ;Y coordinate (no change)
;

```

GIOS Function Reference

```

;Lower the boundary pen.
;
        DB      59,4          ;function code - lower boundary pen
;
;Lift the drawing pen.
;
        DB      39,4          ;function code - lift drawing pen
;
;Move to point E (125,125) of the triangle. Remember, this operation
;lowers the pen.
;
        DB      52,4          ;function code - absolute move
        DW      125           ;absolute X coordinate
        DW      125           ;absolute Y coordinate
;
;Draw to point F (175,125) of the triangle.
;
        DB      56,4          ;function code - incremental draw
        DW      50            ;X coordinate (right 50 pixels)
        DW      0             ;Y coordinate (no change)
;
;Draw to point G (150,175) of the triangle.
;
        DB      56,4          ;function code - incremental move
        DW      -20           ;X coordinate (left 20 pixels)
        DW      50            ;Y coordinate (up 50 pixels)
;
;Lift the boundary pen.
;
        DB      58,4          ;function code - lift boundary pen
;
;Terminate the polygonal area fill.
;
        DB      51,4          ;function code - terminate fill
;
;Use the batch function (0,0) to execute all the functions defined in
;the buffers above in the data segments.
;
;Build the batch command buffer.
;
BATCH      DB      0,0        ;Batch function code
BUF_LEN    DW      64         ;command buffer length
CMD_OFFSET DW      CMD_BUF    ;command buffer offset address
CMD_SEG     DW      0         ;allocate a word to store
                                ; command buffer segment address
DATA       ENDS              ;end of data segment
;
;start program segment.
;
CODE        SEGMENT           ;start code segment
        ASSUME CS:CODE, DS:DATA
;
;Initialize data segment.
;

```

```

START:      MOV     AX,DATA      ;set up data segment register
            MOV     DS,AX
;
            MOV     CMD_SEG,AX   ;store segment in command buffer
;
;Set up entry registers to call AGIOS batch function (0,0).
;DS register is already set up.
;
AGIOS:      MOV     AX,4403H     ;I/O control write
            MOV     BX,1        ;console handle
            MOV     CX,8        ;input buffer length
            MOV     DX, OFFSET BATCH ;input buffer offset address
            INT     21H
;
;Return to MS-DOS.
;
            MOV     AH,4CH       ;MS-DOS function 4CH lets
            INT     21H         ; return to MS-DOS gracefully
CODE        ENDS              ;end of code segment
            END     START       ;end of program

```

Lower Boundary Pen (ESC * p v)

This function lowers the polygon boundary pen. If a boundary pen has been enabled, edges of the polygon are drawn with a solid line pattern regardless of the current defined line type. This remains in effect until the boundary pen is lifted.

Input Buffer:

(4,59)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

The boundary pen is not selected by default. You can select or enable the boundary pen using function (4,24).

(See an example on lifting and lowering boundary pen in function (4,58).)

GRAPHICS STATUS (ESC * s)

The GIOS functions in this section let you request graphics status information. There are twelve graphics statuses and each can be requested with the appropriate function. The information is returned in the buffer specified by the segment and offset address given in your application.

This section gives the GIOS functions that return graphics status information. The following graphics status functions are explained in detail in this section:

<u>Function Code</u>	<u>Function</u>
(4,60)	Not Implemented
(4,61)	Read Pen Position
(4,62)	Read Cursor Position
(4,63)	Not Implemented
(4,64)	Read Display Size
(4,65)	Read Graphics Settings
(4,66)	Read Graphics Text Status
(4,67)	Read Zoom Status
(4,68)	Read Relocatable Origin
(4,69)	Read Reset Status
(4,70)	Read Area Shading
(4,71)	Read Dynamic Graphics Capabilities
(4,72)	Set Picture Definition Defaults
(4,73)	Graphics Hard Reset
(4,74)	Read Extended Screen Dimensions

Graphics Status (ESC * s) sequences are described in Appendix D on Graphics Control Functions.

Read Pen Position (ESC * s 2^)

This function returns the current position and the state of the pen.

Input Buffer:

(4,61)

Function Code

((BUFFER))

Segment and offset address of a 3-word buffer for returning pen status position.

First Word = Offset

Second Word = Segment

Outputs:

AX = 0

Successful

AX <> 0

Unsuccessful

BUFFER contains:

(X-COORD)

The X and Y coordinates of the current pen position.

(Y-COORD)

(STATE)

0 = Pen lifted

1 = Pen lowered

Read Cursor Position (ESC * s 3^)

This function returns the current position of the cursor.

Input Buffer:

(4,62)	Function Code
((BUFFER))	Segment and offset address of a 2-word buffer for returning the cursor position.
	First Word = Offset
	Second Word = Segment

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

BUFFER contains:

(X-COORD)	The X and Y coordinates of current cursor position.
(Y-COORD)	

Read Display Size (ESC * s 5^)

This function returns the number of displayable units in the X and Y axes. It also returns the number of units per millimeters in the display.

Input Buffer:

(4,64)	Function Code
((BUFFER))	Segment and offset address of a 6-word buffer for returning the display size and unit.
	First Word = Offset
	Second Word = Segment

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

BUFFER contains:

(X-LWR-LEFT)	Lower left X and Y coordinates of the maximum display size.
(Y-LWR-LEFT)	
(X-UPR-RIGHT)	Upper right X and Y coordinates of the maximum display size.
(Y-UPR-RIGHT)	
(X-MM)	The X and Y dimensions in number of dots per millimeter.
(Y-MM)	

Remarks:

This function lets you scale data for use on graphics devices with varying display area size.

X-MM and Y-MM can be used to determine the Aspect Ratio of the graphic pixels:

Aspect Ratio = Y-MM/X-MM

Read Graphics Settings (ESC * s 6^)

This function returns information about the current active graphics settings.

Input Buffer:

(4,65)	Function Code
((BUFFER))	Segment and offset address of a 16-word buffer for returning graphics settings.
	First Word = Offset
	Second Word = Segment

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

BUFFER contains the following graphics settings in 16 consecutive words.

- | | |
|-----------------------------|-----------------------------------------------------------------------------------------------------------|
| 1. (CLEAR DISPLAY) | 0 = No clear
1 = Paper advance
2 = Clear (total erase)
*3 = Partial clear by area |
| 2. (NUMBER OF PENS) | *1 |
| 3. (COLOR CAPABILITY) | *0 = Black or green
1 = Gray levels |
| 4. (COLOR LEVEL CAPABILITY) | *0 = No color |
| 5. (AREA SHADING) | 0 = No
*1 = Yes |
| 6. RESERVED | 0 |
| 7. RESERVED | 0 |
| 8. (DYNAMIC MODIFICATION) | 0 = No
*1 = Yes |
| 9. (CHARACTER SIZE) | 0 = Fixed
1 = Integer multiples of the basic cell size
*2 = Fractional scale of the basic cell size |

GIOS Function Reference

- | | |
|------------------------------|-------------------------------------------------------------------------------------------|
| 10. (CHARACTER ORIENTATION) | 0 = Fixed
*1 = Multiples of 90 degrees
2 = Multiples of 45 degrees
3 = Any angle |
| 11. (CHARACTER SLANT) | 0 = Fixed
*1 = 26.57 degrees
2 = Any angle |
| 12. (DOT-DASH LINE PATTERNS) | 0 = None
1 = Pre-defined only
*2 = User-defined and predefined |
| 13. (RESERVED) | *0 |
| 14. (RESERVED) | *0 |
| 15. (RESERVED) | *0 |
| 16. (RESERVED) | *0 |

NOTE

* Indicates the value that the HP 150 always returns.

Read Graphics Text Status (ESC * s 7^)

This function returns the current attributes of graphics text.

Input Buffer:

(4,66)	Function Code
((BUFFER))	Segment and offset address of a 5-word buffer for returning graphics attributes.
	First Word = Offset
	Second Word = Segment

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

BUFFER contains:

(X SIZE)	The X,Y dimensions of the character cell.
(Y SIZE)	
(ORIGIN)	The text origin (0-11)
(ORIENTATION)	The text orientation (0, 90, 180, 270)
(SLANT)	The character slant.
	0 = Not slanted
	27 = Slanted

Read Zoom Status (ESC * s 8^)

This function returns the terminal's zoom setting.

Input Buffer:

(4,67)	Function Code
((BUFFER))	Segment and offset address of a 2-word buffer for returning the zoom setting.
	First Word = Offset
	Second Word = Segment

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

BUFFER contains:

(ZOOM SIZE)	1 - 16 (the HP 150 always returns 1).
(ZOOM ON/OFF)	0 = OFF (the HP 150 always returns 0)
	1 = ON

Read Relocatable Origin (ESC * s 9^)

This function returns the current relocatable origin.

Input Buffer:

(4,68)	Function Code
((BUFFER))	Segment and offset address of a 2-word buffer for returning the relocatable origin.
	First Word = Offset
	Second Word = Segment

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

BUFFER contains:

(X-COORD)	The X and Y coordinates of the current relocatable origin.
(Y-COORD)	

Read Reset Status (ESC * s 10[^])

This function returns information on whether the terminal has executed a full reset (or power on) since the last time reset status was checked.

Input Buffer:

(4,69)	Function Code
((BUFFER))	Segment and offset address of a 8-word buffer for returning reset status.
	First Word = Offset
	Second Word = Segment

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

BUFFER contains:

(RESET STATUS)	0 = no full reset since last check
	1 = terminal has been reset

(RESERVED)
(RESERVED)
(RESERVED)
(RESERVED)
(RESERVED)
(RESERVED)
(RESERVED)

Remarks:

This function tells whether or not you need to re-establish terminal settings or images before resuming terminal functions.

You will need to allocate 7 additional words to BUFFER, but they are currently not used.

Read Area Shading (ESC * s 11^)

This function returns information on the terminal's area shading capability.

Input Buffer:

(4,70)	Function Code
((BUFFER))	Segment and offset address of a 3-word buffer for returning shading information.
	First Word = Offset
	Second Word = Segment

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

BUFFER contains:

(CAPABILITIES)	The area shading capabilities (the HP 150 always returns "2" to indicate that the area shading can be a polygon).
(WIDTH)	The area shading pattern size (the HP 150 always returns "8" for WIDTH and "8" for HEIGHT).
(HEIGHT)	

Read Dynamic Graphics Capabilities (ESC * s 12^)

This function returns information on the terminal's dynamic graphics capabilities.

Input Buffer:

(4,71)	Function Code
((BUFFER))	Segment and offset address of a 2-word buffer for the returning dynamic graphics capabilities.
	First Word = Offset
	Second Word = Segment

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

BUFFER contains:

(SELECTIVE-ERASE-CAPABILITIES)	The HP 150 always returns a "1".
(COMPLEMENT-CAPABILITIES)	The HP 150 always returns a "1".

Remarks:

This function returns the ability of the terminal to change selected portions of the display. The HP 150 has selective erase and complement capabilities.

Set Picture Definition Defaults (ESC * m 1 r)

This function sets the picture definition parameters to their default values.

Input Buffer:

(4,72)	Function Code
(RESET LEVEL)	The level of graphics reset. On the HP 150, the value "1" is the only supported level.

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

The picture defaults are:

```

Pen (DOWN)
Line type (1)
User-defined line pattern (SOLID)
User-defined area fill pattern (SOLID)
Boundary pen (OFF)
Drawing mode (SET)
Text size (1)
Text origin (1)
Text slant (OFF)
Text orientation (1)
Graphics text (OFF)
Area fill type (2)
    
```

Remarks:

Function (4,38) lets you set all the graphics parameters to their default values.

Graphics Hard Reset (ESC * w r)

This function sets the graphics parameters to their power-on state.

Input Buffer:

(4,73)	Function Code
--------	---------------

Outputs:

AX = 0	Successful
AX <> 0	Unsuccessful

Remarks:

This function sets all graphics parameters to their default values plus the following:

1. Clears raster memory buffer
2. Drawing pen is positioned at location (0,0)
3. Sets the graphics character set to the system default character set

Graphics defaults are listed in function (4,38) which sets graphics parameters to their default values.

Read Extended Screen Dimensions

This function provides information about the alphanumeric and graphics screen size plus the relationship between the two.

Input Buffer:

(4,74)	Function Code
((BUFFER))	Segment and offset address of a 10-word buffer for returning screen dimensions.
	First Word = Offset
	Second Word = Segment

Outputs:

AX = 0	Successful
AX = 0	Successful

BUFFER contains:

Graphics screen size, in pixels:

(X-PIXELS)	512 in the horizontal direction
(Y-PIXELS)	390 in the vertical direction

Alphanumeric screen size, in rows and columns:

(ROWS)	24 rows. (24 is returned, but 3 more rows need to be added for softkey labels and status line.)
(COLUMNS)	80 columns

Graphics screen size, in millimeters:

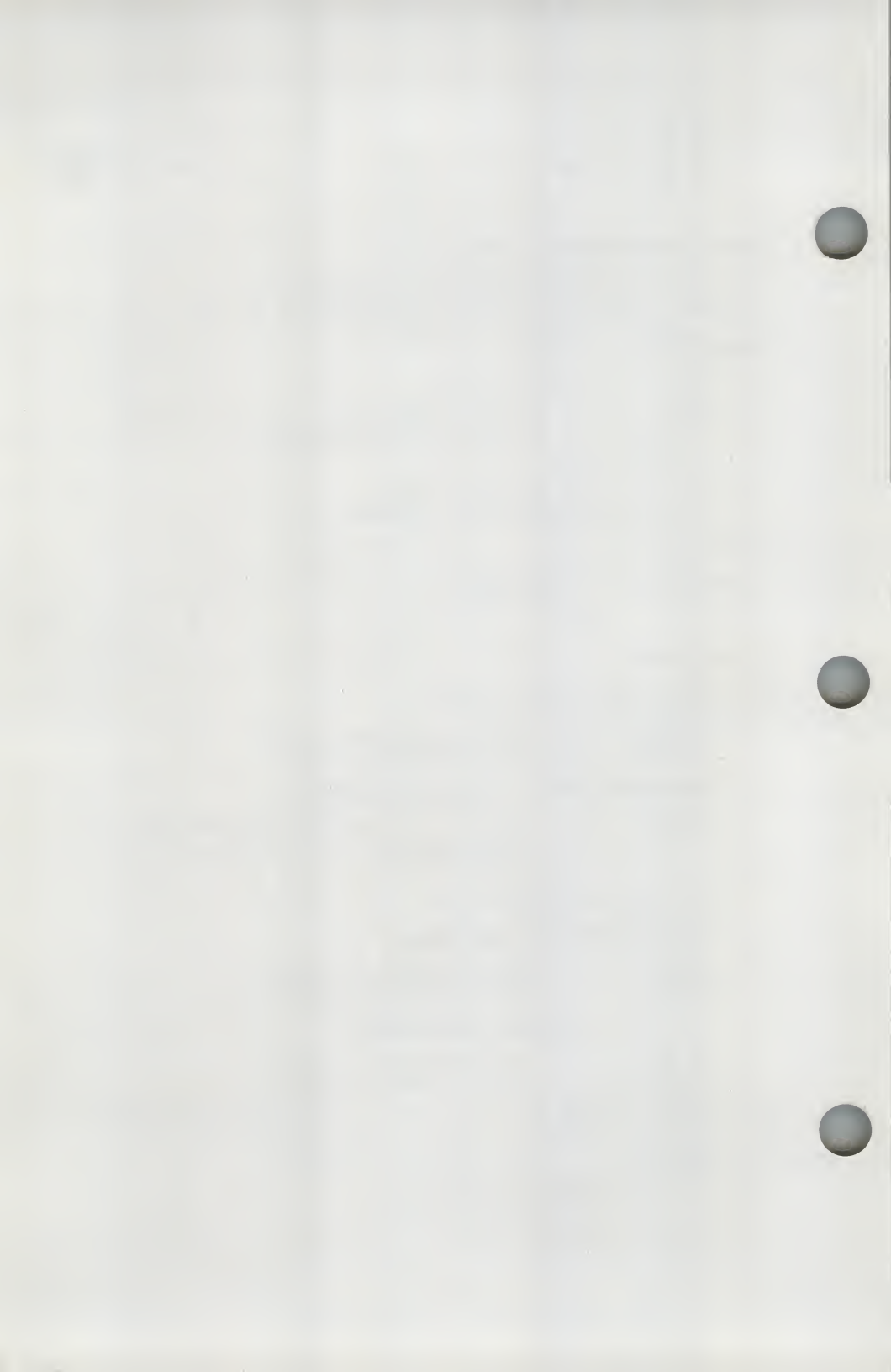
(X-MM)	160 in the horizontal direction
(Y-MM)	120 in the vertical direction

Alphanumeric screen size, in millimeters:

(ROW-MM)	150 in the horizontal direction
(COL-MM)	116 in the vertical direction

Graphics origin minus alphanumeric origin, in millimeters:

(DELTA-X)	10 in the horizontal direction
(DELTA-Y)	4 in the vertical direction



DATA COMMUNICATIONS PROGRAMMING

SECTION

6

NELSYS THT - EL

INTRODUCTION

The HP 150 provides a set of Data Communications (data comm) I/O functions in addition to the standard I/O functions available in MS-DOS. This section discusses how to write your own data comm programs using the standard MS-DOS functions and the HP extended functions. At the end of this section, there is a program that uses these data comm functions to read and write data to the data comm ports in the HP 150. The program is written in Microsoft compiled BASIC and it calls an assembly routine to perform the data comm operations.

WRITING YOUR OWN DATA COMM PROGRAM

The HP 150 contains two serial ports, labeled Port 1 and Port 2. To perform Input/Output operations on the two serial ports, the HP 150 uses data comm devices COM1 and COM2. This may be different from other computers that use hardware port addresses.

You can write data comm programs using devices COM1 and COM2. The steps are listed below:

1. Open device COM1 or COM2 using MS-DOS function 3D hex
2. Read data from the opened device using MS-DOS function 3F hex
3. Write data to the opened device using MS-DOS function 40 hex
4. Perform special operations using MS-DOS I/O function 44 hex

MS-DOS functions 3D hex, 3F hex, 40 hex, and 44 hex are described in the *MS-DOS Programmer's Reference Manual*.

NOTE

The above procedure lets you perform I/O operations on serial ports. For information on performing HP-IB I/O operations, see the *HP 150 Technical Reference Manual* (Product No. 45625A).

ASSIGNING COM1 AND COM2 DEVICES

The HP 150 default device assignments for COM1 and COM2 are Port 1 and Port 2, respectively. The configuration information defined in Port 1 and Port 2 configuration menus are still in effect. Therefore, your application program does not have to set baud-rates, parity, and other data comm parameters. The HP 150 also handles Enq/Ack or Xon/Xoff handshaking if it is selected, and maintains a 256-byte buffer for incoming data on each serial port; therefore, there is less chance for your application to lose data.

You can change COM1 and COM2 device assignments in two ways. First, you can use the Device Config program under P.A.M. to map either COM1 or COM2 to Port 1 or Port 2 or 'Remote'. If COM1 or COM2 is mapped to 'Remote', that device is tied to the Remote/Serial field in the Global Configuration menu. Second, the Remote/Serial field lets you map either Port 1 or Port 2 to 'Remote' or 'Serial'. 'Remote' is typically a mainframe computer and 'Serial' is typically a line printer. You can change these assignments as necessary by following the procedures in "Configuring MS-DOS" in the *HP 150 Personal Computer Owner's Guide*.

ACCESSING SPECIAL DATA COMM FUNCTIONS

In addition to the standard MS-DOS read and write capabilities, there are some additional functions that you can use to control data comm operations. To access a data comm function, you have to set up an input buffer. The first word in the buffer must be the function code for the desired function. If the function requires any additional parameters, they must be stored in the buffer in the order that is indicated in the function description.

Once you have set up the input buffer, you can use MS-DOS I/O control function 44H to execute the data comm control function. In order for function 44H to execute properly, you have to set up the required assembly registers as listed below:

- AX = I/O Control Write (4403H)
- BX = Data comm device handle
- CX = Number of bytes in the input buffer
- DS = Input buffer segment address
- DX = Input buffer offset address

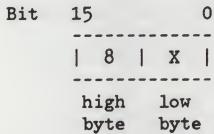
The AX register tells the operating system to perform an I/O control write operation. The BX register has the device handle. When the operating system opens a device, it returns a device handle that is used to identify the device in subsequent system function calls. The CX register has the number of bytes in the buffer. The buffer is addressed by DS and DX registers.

Once your application sets up the above registers, it transfers control to the operating system by issuing an 'INT 21H' instruction. The operating system always returns a completion status in the carry-flag. If "carry" is clear, then the function is successful; otherwise, it is not successful.

The *MS-DOS Programmer's Reference Manual* has additional information on I/O function 44H.

DATA COMM FUNCTION REFERENCE

Each Data Comm function is identified by a 16-bit function code as shown below:



The operating system uses the high byte of this function code to identify the type of control function to perform. For data comm functions, the high byte of the function code is always the value 8. The low byte of the function code is a function number that tells the operating system which data comm function to execute. Your application must store this 2-byte function code in the first word of an input buffer to be passed to the operating system.

In this section, a data comm function code is given in decimal and is represented by an ordered-pair (8,X). You should note that the Intel processor interprets the two bytes in a word in reverse order; that is, in physical memory, the value X is stored before the value 8. There are two ways of defining a function code; you can define it either in byte-mode or in word-mode using the Micro-Soft Macro-Assembler directives Define Byte (DB) or Define Word (DW).

For example, to allocate a buffer called IN_BUF and store the function code (8,1) in the first word of IN_BUF, you can use either one of the following instructions:

For example, to allocate a buffer called IN_BUF and store the function code (8,1) in the first word of IN_BUF, you can use either one of the following instructions:

In Byte-mode:	IN_BUF	DB	1,8
In Word-mode:	IN_BUF	DW	0801H

The above two instructions both allocate a buffer called IN_BUF and store the value 1 in the first byte and the value 8 in the second byte of the buffer.

In order to clarify the parameters your application needs to supply, a standard notation is used in the function reference. A single-parenthesis notation indicates a 16-bit parameter and a double-parenthesis notation indicates a double-word parameter.

The following data comm functions are described in detail in this section:

Function Code	Function
(8,1)	Set 7-bit Mode
(8,2)	Set 8-bit Mode
(8,3)	Enable Transparency Mode
(8,4)	Disable Transparency Mode
(8,5)	Disconnect Modem
(8,6)	Send Break
(8,7)	Send Data Block

Set 7-Bit Mode

This function turns on 7-bit mode. In this mode, your application works with the first 7 bits of each data comm byte. The system firmware assumes the eighth bit is the parity bit and interprets it according to the Parity setting in the Port1 or Port2 Configuration menu. Any selected handshaking is still active. This is the default (power-on) mode of operation for data comm devices.

Input Buffer:

(8,1)	Function Code (1 word)
	Most significant byte = 8
	Least significant byte = 1

Outputs:

Carry = 0	Successful
Carry = 1	Unsuccessful

Example:

The following assembly program segment sets up the required parameters and calls MS-DOS to turn on 7-bit mode for COM1

```
;
;Build an input buffer with the correct function code stored
; in it.
;
IN_BUF   DB   1,8           ;function code to turn on 7-bit mode
COM1_HAN DW   ?           ;allocate space to store device handle
.
.
.

;Set up the required registers. The input buffer's segment
;address is assumed to be the same as the current DS register.
;The COM1 device handle is already stored in COM1_HAN.
;
MOV AX,4403H           ;I/O control write
MOV BX,COM1_HAN        ;BX = COM1 handle
MOV CX,2               ;2 bytes in the buffer
MOV DX,OFFSET IN_BUF   ;pass buffer offset address
INT 21H                ;transfer control to MS-DOS
RET                     ;return
```

Set 8-Bit Mode

This function turns on 8-bit mode. In this mode, your application works with all eight bits of each data comm byte. Any selected handshaking is still active. Depending on the current settings in the Port1 or Port2 Configuration menu, control codes (such as DC1, ACK, ENQ, NULL, DEL, etc.) may be interpreted by the system firmware.

Input Buffer:

(8,2)

Function Code (1 word)

Most significant byte = 8

Least significant byte = 2

Outputs:

Carry = 0

Successful

Carry = 1

Unsuccessful

Enable Data Comm Transparency Mode

This function sets data comm transparency mode. In this mode, handshaking is disabled and all characters are passed directly to your application. The HP 150 system firmware does not process data comm characters.

Input Buffer:

(8,3)

Function Code (1 word)

Most significant byte = 8

Least significant byte = 3

Outputs:

Carry = 0

Successful

Carry = 1

Unsuccessful

Remarks:

Even though handshaking is disabled in transparency mode, the HP 150 processes Xon/Xoff handshaking if it is enabled.

You can specify this function independent of 7-bit mode or 8-bit mode functions, (8,1) and (8,2), respectively.

Disable Data Comm Transparency Mode

This function disables data comm transparency mode. In this mode, the handshake options selected in Port1 or Port2 configuration menus are active. The system firmware monitors the data coming in and out of the selected data comm device. Control characters (such as DC1, ACK, ENQ, NULL, and DEL) are interpreted and absorbed by the system firmware.

Input Buffer:

(8,4)

Function Code (1 word)

Most significant byte = 8

Least significant byte = 4

Outputs:

Carry = 0

Successful

Carry = 1

Unsuccessful

Remarks:

This is the default mode of operation for Data Comm devices.

Disconnect Modem

This function drops the Device Terminal Ready (DTR) hardware signal which causes a modem disconnect.

Input Buffer:

(8,5)

Function Code (1 word)

Most significant byte = 8

Least significant byte = 5

Outputs:

Carry = 0

Successful

Carry = 1

Unsuccessful

Remarks:

This function is designed to disconnect a modem from an application running under MS-DOS. The escape sequence ESC f disconnects the modem in Terminal mode; it does not work in Computer mode.

Send Break

This function is equivalent to typing the Break key from the keyboard in Terminal mode. The system firmware does it by holding the hardware transmit line low for 200 milliseconds.

Input Buffer:

(8,6)

Function Code (1 word)

Most significant byte = 8

Least significant byte = 6

Outputs:

Carry = 0

Successful

Carry = 1

Unsuccessful

Remarks:

This function is designed to send a 'break' to your application programmatically. The Break key on the keyboard only works in Terminal mode; it does not work in Computer mode. That is, you cannot use the Break key to abort an application currently running under MS-DOS.

Read/Write Data Comm Blocks

The HP 150 system firmware, while offering a very high level of functionality, can also operate at high speed if Input and Output operations are performed on blocks of characters instead of one character at a time. When using data comm, there are special multiple-character read and write functions that can significantly improve performance.

For block output, an application can use function (8,7) to write a block of data to a data comm port. Function (8,7) is described in detail in the next section.

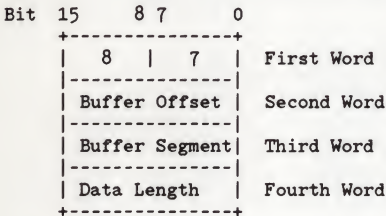
Send Data Block

This function sends one block of data at a time to the selected data comm device rather than transmitting one byte at a time. Data is stored in a buffer specified by your application. There is no system limit on the buffer size, except that the buffer length should not be too big to fit in the DATA LENGTH parameter.

Input Buffer:

(8,7)	Function Code (1 word)
((BUFFER))	An address that points to the data buffer (2 words). First Word = Offset address Second Word = Segment address
(DATA LENGTH)	A word that contains the number of bytes in the data buffer.

The following diagram shows how the input buffer should look:



Outputs:

Carry = 0	Successful
Carry = 1	Unsuccessful

Remarks:

This function is similar to the standard MS-DOS Write function (40H) except that it takes less time to send a block of data.

***** See an example of this function on the next page *****

Data Communications Programming

Example:

The following assembly program segment calls function (8,7) to send the string 'HELLO' to device COM1. It assumes that device COM1 is already opened and the device handle is stored in COM1_HAN.

```
;
;Build an input buffer with the required parameters stored in it.
;The parameters must be stored in the order that is specified in
;the function description.
;
IN_BUF    DW    0807H        ;send data block function code
          DW    DATA_OFF    ;the offset address of data block
DATA_SEG  DW    ?           ;the segment address of data block
          ; (to be assigned)
          DW    5            ;5 bytes in data block
DATA_OFF  DB    'HELLO'     ;define the string
COM1_HAN  DW    ?           ;allocate space to store handle
          ;(value to be assigned)
          .
          .
          .
;
;Ready to send the string 'HELLO'. Set up the required
;register. The input buffer's segment address is assumed to be
;the same as the current DS.
;
MOV AX,DS        ;assume data block segment address
MOV DATA_SEG,AX ; be the same as the current DS
MOV AX,4403H     ;I/O control write
MOV BX,COM1_HAN  ;move COM1 handle into BX
MOV CX,8         ;8 bytes in input buffer
MOV DX,OFFSET IN_BUF ;input buffer offset
INT 21H          ;call MS-DOS to send the data
```

Read Data Block

For block input, an application can use MS-DOS I/O control read function 44H to get data into an application buffer. Once an application reads a block of data into a buffer, it can read single characters from the buffer. When the application buffer is empty, the application makes another "I/O control read" request to the system firmware. Since the system firmware maintains a 256-byte buffer for each data comm port, it can be buffering up additional characters while the application is working on its own buffer characters. This technique improves an application's data comm performance significantly.

Data comm functions (8,1) through (8,7), described earlier, use I/O control sub-function 3 (4403H) to perform a "write to control channel" operation. To read a block of data, you can use sub-function 2. I/O control sub-function 2 (4402H) lets you read as many bytes as the system firmware has buffered without waiting for an exact number of characters. Since the system maintains a 256-byte buffer, an application should allocate a buffer of the same size or less for storing data.

Unlike the I/O control sub-function 3, sub-function 2 does not use an input buffer stored with the function code and other parameters; it only requires a buffer to return the data block.

Input Buffer:

((BUFFER))

A pointer to a 2-word buffer for returning the data block

First word = Buffer offset address
Second word = Buffer segment address

Outputs:

Carry = 0

Successful

Carry = 1

Unsuccessful

AX

Number of bytes returned in the specified buffer

*** See an example of this function on next page ***

Data Communications Programming

Example:

The following program segment reads a block of data into a buffer named DC_BUF.

```
;
;Allocate a buffer to read a data block.
;
DC_BUF    DB    256 DUP(?)           ;allocate 256 bytes of
                                        ; uninitialized buffer
DC_COUNT  DW    ?                   ;stores DC_BUF byte count
COM1_HAN  DW    ?                   ;stores device handle
.
.
.
;
;Assumed that the device is already opened and the device handle is
;stored in COM1_HAN.
;
    MOV  AX,4402H                   ;I/O control read
    MOV  BX,COM1_HAN                ;get handle from COM1_HAN and
                                        ; store in BX
    MOV  CX,256                     ;read up to 256 bytes
    MOV  DX,OFFSET DC_BUF           ;pass buffer offset address,
                                        ;assume segment address already
                                        ; in DS
    INT  21H                        ;call MS-DOS to read data
    MOV  DC_COUNT,AX                ;save byte count in DC_COUNT
    RET                             ;return
```

Programming Considerations

MS-DOS has two processing modes: Cooked mode and Raw mode. If a device is in Cooked mode, MS-DOS checks each character passing through the device for special significance. If a device is in Raw mode, MS-DOS passes characters directly to the device; it does not try to interpret the characters.

When you open data comm device COM1 or COM2, it is normally in Cooked mode. In Cooked mode, MS-DOS checks for characters such as an end-of-file marker, printer echo, and other special characters, and processes them.

By turning the data comm device into Raw mode, you can prevent MS-DOS from looking for any special characters. As a result, you can speed up the data comm performance. You can use MS-DOS function 44 hex to set the data comm device in Raw mode; this function is presented in the *MS-DOS Programmer's Reference Manual*.

Data Communications Programming

The following assembly program segment opens device COM1 and enters RAW mode:

```
COM1_HAN DW ? ;allocate space to hold COM1 handle
COM1_DEV DB 'COM1',0 ;allocate a buffer with the device
; name stored in it. Zero
; terminates the name.
.
.
.
;
;Open device COM1.
;
MOV DX,OFFSET COM1_DEV ;points to device name to open
MOV AH,3DH ;MS-DOS open file function code
MOV CX,2 ;request read/write access
INT 21H ;call MS-DOS to execute function
JC ERROR ;if error occurs, carry is set
MOV COM1_HAN,AX ;handle return in AX; save it
;
;Set Raw mode to disable MS-DOS processing.
;
MOV BX,COM1_HAN ;put COM1 handle in BX
MOV AX,4400H ;I/O control - get device status
INT 21H ;call MS-DOS to get device status
MOV DH,0 ;clear high byte of status word
; returned in DX
OR DL,20H ;set Raw bit (bit 5)
MOV BX,COM1_HAN ;put COM1 handle in BX
MOV AX,4401H ;I/O control - set device status
INT 21H ;call MS-DOS to turn on Raw mode
RET ;return
```

AN EXAMPLE OF A DATA COMM PROGRAM

The following BASIC program calls an assembly routine to copy 80 characters from Port 2 to Port 1.

```
5 defint a-z
10 dim o(80)
11 for x=1 to 80
12 o(x)=x
13 next x
15 print "before open"
20 call popen
22 print "before write"
25 call ptwr
27 print "before read"
30 call pord(o(0))
35 print "ready to print values"
37 print "number of characters";o(0)
40 for x=1 to 80
50 print o(x);
60 next x
65 call pclose
70 end
```

Data Communications Programming

```

DATA      SEGMENT WORD PUBLIC 'DATA'
BUF_OUT DB      'ABCDEFGH',0,'I',70 DUP (0)
BUF_IN1 DB      13,10
BUF_IN DB      80 DUP (?),10,13,'$'
PORT_1 DB      'COM1',0,0,0,0
PORT_2 DB      'COM2',0,0,0,0
HAND_1 DW      1
HAND_2 DW      1
KP_PRT DW 1
CHG_RAW DW ?
CHG_TRN DB 3,8
CHG_NRM DB 4,8
DATA      ENDS

```

DGROUP GROUP DATA

```

CODE      SEGMENT BYTE PUBLIC 'CODE'
ASSUME CS:CODE,DS:DGROUP

PUBLIC POPEN,PTWR,FORD,PCLOSE

```

```

POPEN     PROC FAR
          PUSH BP
          MOV BP,SP
          PUSH DS

          MOV AX,3D02H                ;open Port1
          LEA DX,PORT_1
          INT 21H
          MOV HAND_1,AX

          MOV AX,3D02H                ;open Port2
          LEA DX,PORT_2
          INT 21H
          MOV HAND_2,AX

          MOV AX,4400H                ;get COM1 characteristics
          LEA DX,CHG_RAW
          XOR CX,CX
          MOV BX,HAND_1
          INT 21H

          MOV KP_PRT,DX

          OR DX,0020H                 ;SET TO RAW MODE
          XOR DH,DH
          MOV AX,4401H
          INT 21H

          MOV AX,4403H                ;set up transparency mode
          LEA DX,CHG_TRN
          MOV CX,0002H

```

```

        MOV BX,HAND_1
        INT 21H

        POP DS
        POP BP
        RET 0
POPEN  ENDP

PTWR   PROC FAR
        PUSH BP
        MOV BP,SP
        PUSH DS

        MOV AX,4000H
        LEA DX,BUF_OUT           ;output data to Port2
        MOV CX,80
        MOV BX,HAND_2
        INT 21H

        POP DS
        POP BP
        RET 0
PTWR   ENDP

PORD   PROC FAR
        PUSH BP
        MOV BP,SP
        PUSH DS

        MOV AX,4402H
        MOV BX,HAND_1           ;input data from Port1
        LEA DX,BUF_IN
        MOV CX,80
        INT 21H

        MOV CX,AX
        XOR SI,SI
        XOR AX,AX
        MOV BX,[BP+6]
        MOV [BX],CX

TRNLP: MOV AL,BUF_IN[SI]
        MOV [BX]+2,AX
        ADD BX,2
        INC SI
        LOOP TRNLP

        POP DS
        POP BP
        RET 2
PORD   ENDP

PCLOSE PROC FAR
        ;set the ports back to normal and
        ;close them

```

Data Communications Programming

```
PUSH BP
MOV BP,SP
PUSH DS
```

```
MOV AX,4403H           ;turn off transparency mode
LEA DX,CHG_NRM
MOV CX,0002H
MOV BX,HAND_1
INT 21H
```

```
MOV DX,KP_PRT          ;turn on Cook mode
XOR DH,DH
MOV AX,4401H
XOR CX,CX
MOV BX,HAND_1
INT 21H
```

```
MOV AX,3E00H            ;close Port 1
MOV BX,HAND_1
INT 21H
```

```
MOV AX,3E00H            ;close Port 2
MOV BX,HAND_2
INT 21H
```

```
POP DS                  ;return
POP BP
```

```
RET 0
PCLOSE ENDP
```

```
CODE ENDS
END
```

AGIOS AND ESCAPE SEQUENCE QUICK REFERENCE

APPENDIX

A

This Appendix is intended to give you a quick reference to the AGIOS function calls and the equivalent escape sequences, where applicable. You need to refer to AGIOS sections to determine the parameters values which correspond to each of these function calls.

AGIOS Call	Description	Equivalent Escape Sequence
---------------	-------------	-------------------------------

(0,0)	Batch	
-------	-------	--

AIOS - Video Intrinsics

(0,1)	Define Area
(0,2)	Write Area
(0,3)	Clear Area
(0,4)	Enhance Area
(0,5)	Read Area
(0,6)	Shift Area
(0,7)	Write Line

AIOS - Application Softkeys

(0,8)	Update Softkey Label
(0,9)	Read Softkey Label
(0,11)	Display Softkey Labels

(0,12) - (0,15) Reserved

AIOS - Control Functions

(0,16)	Execute Two-Character Sequence	ESC character
(0,17)	Position Cursor	ESC & a
(0,18)	Define Enhancements	ESC & d
(0,19)	Cursor Sense Absolute	ESC a
(0,20)	Cursor Sense Relative	ESC `
(0,21)	Set Cursor Type	

(0,22) - (0,24) Reserved

(0,25)	Read Fast AGIOS Entry Address
(0,26)	Go to Terminal Mode

(0,27) - (0,31) Reserved

AGIOS and Escape Sequence Quick Reference

AGIOS Call	Description	Equivalent Escape Sequence
---------------	-------------	-------------------------------

AIOS - Touchscreen Functions

(0,32)	Define Touch Field	ESC - z g
(0,33)	Define Softkey Field	ESC - z s
(0,34)	Delete Touch Field	ESC - z d
(0,35)	Touchscreen Reset	ESC - z j
(0,36)	Set Touch Sensing Modes	ESC - z n

(0,37) - (0,39) Reserved

AIOS - Keyboard Processing

(0,40)	Define Key Characteristics
(0,41)	Get Key Characteristics
(0,42)	Put Key
(0,43)	Turn Keycode Mode On/Off
(0,44)	Read Keycode Mode Status
(0,45)	Read Numeric/Graphics Keypad Status
(0,46)	Read Key

(0,47) - (0,79) Reserved

AGIOS and Escape Sequence Quick Reference

AGIOS Call	Description	Equivalent Escape Sequence
---------------	-------------	-------------------------------

GIOS - Display Control (ESC * d)

(4,1)	Clear Graphics Memory	ESC * d a
(4,2)	Set Graphics Memory	ESC * d b
(4,3)	Turn On Graphics Display	ESC * d c
(4,4)	Turn Off Graphics Display	ESC * d d
(4,5)	Turn On Alphanumeric Display	ESC * d e
(4,6)	Turn Off Alphanumeric Display	ESC * d f
(4,7)	Turn On Graphics Cursor	ESC * d k
(4,8)	Turn Off Graphics Cursor	ESC * d l
(4,9)	Turn On Rubber Band Line	ESC * d m
(4,10)	Turn Off Rubber Band Line	ESC * d n
(4,11)	Move Graphics Cursor Absolute	ESC * d <x>,<y> o
(4,12)	Move Graphics Cursor Relative	ESC * d <x>,<y> p
(4,13)	Turn On Alphanumeric Cursor	ESC * d q
(4,14)	Turn Off Alphanumeric Cursor	ESC * d r
(4,15)	Turn On Graphics Text Mode	ESC * d s
(4,16)	Turn Off Graphics Text Mode	ESC * d t

GIOS - Vector Drawing Mode (ESC * m)

(4,17)	Select Drawing Mode	ESC * m <mode> a
(4,18)	Select Line Type	ESC * m <type> b
(4,19)	Define Line Pattern and Scale	ESC * m <pattern> <scale> c
(4,20)	Define Area Fill Pattern	ESC * m <pattern> d
(4,21)	Fill Rectangular Area, Absolute	ESC * m <x1>,<y1> <x2>,<y2> e
(4,22)	Fill Rectangular Area, Relocatable	ESC * m <x1>,<y1> <x2>,<y2> f
(4,23)	Select Polygonal Fill Pattern	ESC * m <pattern> g
(4,24)	Select Boundary Pen	ESC * m <pen> h
(4,25)	No Polygon Boundary	ESC * m h
(4,26)	Set Relocatable Origin	ESC * m <x,y> j
(4,27)	Set Relocatable Origin to Pen Position	ESC * m k

AGIOS and Escape Sequence Quick Reference

AGIOS Call	Description	Equivalent Escape Sequence
(4,28)	Set Relocatable Origin to Cursor Position	ESC * m l
(4,29)	Set Graphics Text Size	ESC * m <size> m
(4,30)	Set Graphics Text Orientation	ESC * m <orientation> n
(4,31)	Turn On Text Slant	ESC * m o
(4,32)	Turn Off Text Slant	ESC * m p
(4,33)	Set Graphics Text Origin	ESC * m <origin> q
(4,34)	Graphics Text Label	ESC * l <text>
(4,35)	Define User Character Set	
(4,36)	Select Default Character Set	
(4,37)	Output Single Text Character	
(4,38)	Set Graphics Default	ESC * m r
(4,72)	Set Picture Definition Defaults	ESC * m l r
(4,73)	Graphics Hard Reset	ESC * w r

GIOS - Graphics Plotting

(4,39)	Lift Pen	ESC * p a
	Vector Move:	ESC * p a <x>,<y>
(4,40)	absolute	
(4,41)	incremental	
(4,42)	relocatable	
(4,43)	Lower Pen	ESC * p b
	Vector Draw:	ESC * p b <x>,<y>
(4,44)	absolute	
(4,45)	incremental	
(4,46)	relocatable	
(4,47)	Set Pen Position to Cursor Position	ESC * p c
(4,48)	Point Plot	ESC * p d
(4,49)	Set Relocatable Origin to Pen Position	ESC * p e
(4,50)	Start Polygonal Area Fill	ESC * p s
(4,51)	Terminate Polygonal Area Fill	ESC * p t
	Polygon Move:	
(4,52)	absolute	
(4,53)	incremental	
(4,54)	relocatable	
	Polygon Draw:	
(4,55)	absolute	
(4,56)	incremental	
(4,57)	relocatable	
(4,58)	Lift Boundary Pen	ESC * p u
(4,59)	Lower Boundary Pen	ESC * p v

AGIOS and Escape Sequence Quick Reference

AGIOS Call	Description	Equivalent Escape Sequence
---------------	-------------	-------------------------------

GIOS - Graphics Status

(4,60)	Not Implemented	
(4,61)	Read Pen Position	ESC * s 2^
(4,62)	Read Cursor Position	ESC * s 3^
(4,63)	Not Implemented	
(4,64)	Read Display Size	ESC * s 5^
(4,65)	Read Graphics Settings	ESC * s 6^
(4,66)	Read Graphics Text Status	ESC * s 7^
(4,67)	Read Zoom Status	ESC * s 8^
(4,68)	Read Relocatable Origin	ESC * s 9^
(4,69)	Read Reset Status	ESC * s 10^
(4,70)	Read Area Shading	ESC * s 11^
(4,71)	Read Dynamic Graphics Capabilities	ESC * s 12^
(4,74)	Read Extended Screen Dimensions	

AGIOS FUNCTION CALLS FROM HIGH-LEVEL LANGUAGES

APPENDIX

B

NELSYS THT - EL

The purpose of this appendix is to give you examples on how to use AGIOS function calls in high-level languages. Although the language example used here is in MS-Pascal, the same principles apply to other high-level languages. The example calls an assembly routine to home the cursor. Even though the operation is a simple one, you can apply the concepts to call other AGIOS functions. A more complex MS-BASIC program calling AGIOS touch functions is also included at the end of this appendix.

NOTE

If you are using a different high-level language, you should understand the concepts discussed in this section. Then you should read the section in the language manual that discusses calling an assembly routine and passing parameters to it for the actual details of implementation.

GETTING STARTED WITH PASCAL

To use AGIOS function calls, you need a working knowledge of the following areas:

- o HP 150 capabilities (touchscreen, graphics, etc.)
- o AGIOS function calls
- o Intel 8086/8088 assembly language
- o MS-DOS system calls
- o MS-Pascal

You need these reference materials handy:

- o The MS-Pascal language manual
- o The MS-Pascal compiler manual
- o The MS-DOS Programmer's Reference Manual
- o An 8086/8088 assembly language reference manual

You should have the following software:

- o The MS-Pascal compiler and linker
- o The MS-8088 macro assembler
- o A text editor or word processor to create source code files

The first thing you need to do is to create a buffer that contains the AGIOS function code and other information that may be required for the function call. The following example homes the cursor by using the AGIOS function (0,16), "Execute Two-Character Sequence". The buffer is set up as a PACKED ARRAY OF CHAR, with the value 16 in byte 0, the value 0 in byte 1, and the character H in byte 2. This is because the Intel 8088 processor addresses information in a word in reverse order (low byte first, then high byte). The data is passed as follows:

Contents-->	16	0	H
Byte -->	0	+1	+2

Since the data buffer has three bytes, we need to pass a length of 3 to the assembly language program.

Here is an example of a simple Pascal program that homes the cursor:

```
program try_agios (input,output);
```

```
{This program demonstrates the passing of input
parameters to an assembly language routine. The program
homes the cursor.}
```

```
{define an agios data buffer type}
```

```
type    buffer = record
    low_byte   : byte;
    high_byte  : byte;
    char_code   : char
end;
```

```
{declare assembly language routine as external; pass the
data by reference and the length by value. Pick up the
AGIOS error status as an integer result of the function
(0 = no error).}
```

AGIOS Function Calls from High-Level Languages

```
function agios( var  parms : buffer;
                length : integer) : integer; extern;

{declare variables}

var  homeup : buffer;
     ret    : integer;

begin

    {load the agios input values into the buffer}

    homeup.low_byte := 16;
    homeup.high_byte := 0;
    homeup.char_code := 'H';

    {execute agios homeup function and get error status}

    ret  := agios(homeup,3);

    {write return status; zero is no error}

    writeln('return status: ',error);

end.
```

NOTE

MS-Pascal has the pre-defined type BYTE. See the MS-Pascal language reference manual for more information.

It may seem like a lot of work to just home the cursor, but the AGIOS method provides faster I/O. There are advantages to using more powerful AGIOS functions that write large blocks of text to the screen and handle graphics output.

GETTING AGIOS PARAMETERS FROM THE STACK

When the Pascal program passes control to the assembly routine, the parameters are pushed on the stack first. The Pascal program then pushes the return address on the stack. The stack grows downward and the stack pointer (SP) always points to the last element pushed onto the stack. The assembly routine must know how to get the input parameter from the stack.

You should read the section in the MS-Pascal compiler manual that discusses passing parameters. Here are a few key points to remember:

- o Pascal only does FAR calls. Each call puts a return segment address and a return offset address on the stack.
- o Pascal pushes the parameters on the stack from left to right. That is, the leftmost parameter in the argument list is pushed on the stack first and the rightmost parameter is pushed last.
- o Pascal expects the assembly routine to clean up the passing parameters on the stack. You can clean up space allocated on the stack for the parameters by using the assembler instruction RET <n>. <n> is 2 times the number of parameters passed.

To demonstrate the use of the stack in passing parameters, let us assume the following Pascal call statement:

```
AGIOS (parameter1, parameter2)
```

Since Pascal pushes the parameters on the stack from left to right, the following table illustrates the state of the stack at the time the CALL statement is executed:

High Memory	+-----+	
	Parameter1	SP + 6

Stack Pointer	Parameter2	SP + 4

	Return Segment	SP + 2

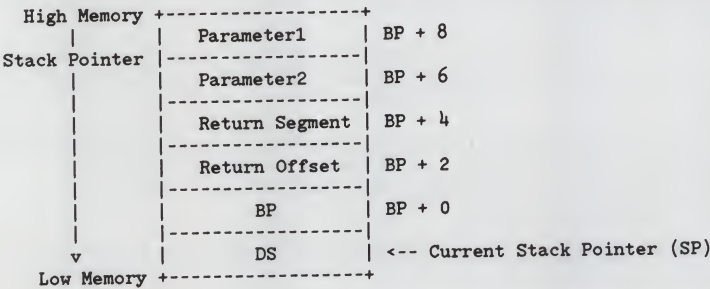
↓		
Low Memory	Return Offset	<-- Current Stack Pointer (SP)
	+-----+	

AGIOS Function Calls from High-Level Languages

In order to fetch the parameters from the stack, it is recommended that you use the BP (Base Pointer) register as the index pointer. Before assigning BP as the index pointer, you should preserve the current BP value. You can save BP by pushing it on the stack. After BP is saved, you can then assign BP to the current SP. You should also save DS because it contains the parameters' data segment address. You can save DS by pushing it on the stack. The following assembly instructions implement the above procedure:

```
PUSH BP
MOV BP,DS
PUSH DS
```

As a result, the state of the stack looks as follows (with BP equates to SP):



The assembly program on the next page shows how to fetch the parameters from the stack using BP as the base index pointer.

ASSEMBLY LANGUAGE INTERFACE TO PASCAL

The next thing the assembly routine has to do is to load certain registers with the information passed to it on the stack. The assembly routine can use the MOV instruction to transfer information pointed by BP plus a positive offset to the assembly registers. For example, to move Parameter2 from the previous example into the CX register, you can use the following assembly instruction:

```
MOV  CX,[BP+6]
```

The following registers are used when you issue an AGIOS call:

<u>Register</u>	<u>Function</u>
AH	Contains the MS-DOS system function number that does device I/O control. This is function number 44H and is described in the <i>MS-DOS Programmer's Reference Manual</i> .
AL	Specifies the I/O control options in function 44H. For AGIOS calls, this option number is 3 and it indicates a "write" to the device control channel.
BX	Contains the device handle number. All AGIOS functions use <u>1</u> as the device handle.
CX	Specifies the number of bytes being passed in the input buffer. In the earlier "home cursor" example, this was 3.
DS	Contains the input buffer's segment address. This address is passed to the routine in the DS register; therefore, the assembly routine should not alter the DS register upon entry from a Pascal program.
DX	Contains the input buffer's offset address. This is part of the address when we pass data by reference

The last step in the assembly language routine is to issue an INT 21H instruction. This instruction generates a software interrupt that passes program control to MS-DOS to perform the AGIOS function.

Upon return from the interrupt, the AX register is loaded with a completion status. A zero value indicates no error while non-zero values indicate errors. The Pascal example in the next section entitled "Passing Pointers to AGIOS from Pascal" shows the AX being returned.

AGIOS Function Calls from High-Level Languages

Here is the assembly routine named AGIOS that homes the cursor:

```
PGROUP    GROUP    PROG
PROG      SEGMENT  BYTE PUBLIC 'PROG'
          ASSUME    CS:PGROUP

          PUBLIC    AGIOS

;
; calling convention: AGIOS(buf,count)
;
; A picture of the stack after the Pascal call.
;
; high memory
; +-----+
; |low byte | high byte| Buffer address
; +-----+
; |low byte | high byte| Buffer length
; +-----+
; |low byte | high byte| Return segment address
; +-----+
; SP-> |low byte | high byte| Return offset address
; +-----+
; low memory
;
;
; stack grows down
;
;
AGIOS     PROC      FAR
PUSH      BP          ;Save callers registers
MOV       BP,SP       ;Assign BP as the index pointer
PUSH      DS
;
```

AGIOS Function Calls from High-Level Languages

```

;
;After BP is pushed to the stack, the SP pointer moves down
;2 bytes. Therefore, the first parameter, buffer length,
;is stored at the location pointed by (SP+6).
;
MOV     DX,[BP+8]      ;Get buffer address
MOV     CX,[BP+6]      ;Get buffer length
MOV     BX,1           ;Console output handle

MOV     AX,4403H       ;Call the I/O control function
INT     21H            ; 44H and use option 3 to write
                        ; to device control channel

POP     DS             ;Restore registers
POP     BP

RET     4              ;Toss 4 bytes, previously
                        ; occupied by agios parameters
                        ; on the stack.

AGIOS   ENDP
PROG    ENDS
        END

```

PASSING POINTERS TO AGIOS FROM PASCAL

Some AGIOS functions require blocks of data that are not in the input buffer to be passed to or from your program. For example, the Write Area function needs a double-word pointer to the buffer which contains the text that you wish to write to the screen. The Cursor Sense Absolute function requires a double-word pointer to a buffer that AGIOS uses to return the current X,Y coordinates.

The double-word pointer has the following structure:

```
Word 1 = Offset address of the data buffer
Word 2 = Segment address of the data buffer
```

The best way to get this information is to use a special data type available in MS-Pascal, ADS. ADS provides the information you need to get the buffer address.

The following example uses the AGIOS function that returns the absolute position of the alpha cursor. The AGIOS input string for this function is the address of the buffer to be used for returning the cursor position.

```
program cursor_sense_absolute (input,output);

type return_buf = array[1..2] of integer;
   input_buf = record
       func_low_byte : byte;
       func_hi_byte  : byte;
       address       : address of return_buf
   end;

var ret      : integer;
    agios_inputs : input_buf;
    cursor_pos  : return_buf;
```

AGIOS Function Calls from High-Level Languages

```
function agios ( var parms : input_buf;  
                 length : integer) : integer; extern;  
  
begin  
  
  {load function code 0,19}  
  agios_inputs.func_low_byte := 19;  
  agios_inputs.func_hi_byte  := 0;  
  
  {load address of output buffer}  
  agios_inputs.address := ads cursor_pos;  
  
  {call agios function}  
  ret := agios(agios_inputs,6);  
  {cursor position is now in the integer array CURSOR_POS}  
  
  writeln('return status =',ret);  
  writeln('row =',cursor_pos[1]);  
  writeln('col =',cursor_pos[2]);  
end.
```

Refer to the MS-Pascal language reference manual for more information on using the ADS type and function.

CALLING AGIOS FROM BASIC

The example in this section shows how to call an AGIOS routine from BASIC. You should read the BASIC compiler manual for details on calling and passing parameters to an assembly routine.

NOTE

Calling an assembly routine from the interpretive BASIC is difficult and cumbersome. Therefore, the compiled BASIC is recommended.

The following are a few key points to remember when calling AGIOS routines from Compiled BASIC:

- o Although the USR function allows access to assembly routines, the CALL statement is recommended for interfacing the Intel 8088 family of assembly language programs with BASIC.
- o BASIC pushes parameters onto the stack from left to right.
- o The assembly routine must know the number of parameters passed. References to parameters are positive offsets added to the BP register (assuming the called routine moved the current stack pointer into the BP register).
- o BASIC passes parameters by reference only; that is, the address of the parameter is passed, not the actual value.
- o The assembly routine must know the variable type for the passed parameter. If possible, pass integer variables to the assembly routine instead of short or long real numbers. Integer variable is stored in two bytes. The address of the low byte (lower order 8 bits) is passed on the stack.
- o If the parameter is a string, the address of the parameter points to a 4-byte string descriptor. Byte 0 and 1 of the string descriptor contains the length of the string. Bytes 2 and 3, respectively, are the offset and segment addresses of the string starting address.
- o If the argument is a string literal in the program, the string descriptor will point to program text. To avoid altering your program, add a null string to the string literal in your program. For example, the statement:

```
10 A$ = "BASIC" + ""
```

will force the string literal to be copied into BASIC string space, not program space. Now the string may be modified without affecting the program.

- o Strings may be altered by assembly routines, but the length must not be changed. BASIC cannot correctly manipulate strings if their lengths are modified by external routines.

AGIOS Function Calls from High-Level Languages

- o The assembly routine must do a RET <n> statement, where <n> is 2 times the number of parameters in the argument list. This statement adjusts the stack to the start of the calling sequence.
- o Value is returned to BASIC by including the variable name that will receive the result in the argument list.

A BASIC PROGRAM CALLING AGIOS FUNCTIONS

```
100 REM This Basic program demonstrates how an AGIOS function can be
110 REM called from Compiled BASIC.
120 REM
130 REM This program calls AGIOS to define some touch fields.
140 REM The steps are listed below:
150 REM 1) Initialize escape sequences and set up definitions.
160 REM 2) Clear the screen, turn off the touch fields and set the
170 REM    reporting mode.
180 REM 3) Define 5 touch fields - these fields will return a single
190 REM    Roman-8 character to a Basic INPUT$ statement
200 REM 4) Read that character and print it on the screen
210 REM
220 WIDTH 255
230 CLS$=CHR$(27)+"H"+CHR$(27)+"J"           'Escape sequence to clear screen
240                                           'Function to move cursor
250 DEF FNLOCATE$(ROW,COL)=CHR$(27)+"&a"+STR$(ROW)+"r"+STR$(COL)+"c"
260 PRINT CLS$                               'Clear the screen
270 CALL OFFTOUCH                             'Initialize touchscreen
280 ROW%=5                                    'Set up parameters for
290 COL%=6                                    '    calls to FNTOUCH -
300 ROWINC%=2                                'Set up row increment
310 COLINC%=10                               'Set up column increment
320 FOR I = 1 TO 5                           'Define 5 fields
330 ROW%=I*4-3                               'Calculate row number
340 RESPONSE$=CHR$(I+48)                    'Calculate response string
350 CALL FNTOUCH(ROW%,COL%,ROWINC%,COLINC%,RESPONSE$) 'Define the field
360 PRINT FNLOCATE$(ROW%+1,7);"Choice ";RESPONSE$ 'Print field label
370 NEXT I                                    'Loop
380 A$=INPUT$(1)                             'Read touch response
390 PRINT FNLOCATE$(20,1);"Choice ";A$;" was selected" 'Echo the response
400 END
```

AGIOS Function Calls from High-Level Languages

```

;*****
;
; FNTOUCH(START_ROW, START_COLUMN, NUM_ROWS, NUM_COLUMNS, RESPONSE)
;
; Define a touch field at (START_ROW, START_COLUMN). The lower
; right corner of the field is calculated by adding NUM_ROWS
; to START_ROW and NUM_COLUMNS to START_COLUMN. The response
; string is pointed to by RESPONSE (the first word is the
; length of the string and the second word is the address of
; the string.
; There is no functional return.
;
;*****

```

```

CODE    SEGMENT PARA PUBLIC 'CODE'
ASSUME  CS:CODE,DS:CODE,ES:NOTHING,SS:NOTHING
        PUBLIC FNTOUCH

START_ROW    EQU    14           ;Offset to parmeters on
START_COLUMN EQU    12           ; the stack
NUM_ROWS     EQU    10
NUM_COLUMNS  EQU    8
RESPONSE     EQU    6

STACK_SIZE   EQU    5*2         ;5 parmeters of 2 bytes each
        page
FNTOUCH PROC FAR

        PUSH    BP               ;Save incoming BP register
        MOV     BP,SP            ;Point to parmeters
        PUSH    DS              ;Save incoming data segment

        MOV     SI,START_ROW[BP] ;Get address of start row
        MOV     AH,[SI]          ;Put row number in high byte

        MOV     SI,START_COLUMN[BP] ;Point to column number
        MOV     AL,[SI]          ;Put column in low byte

        MOV     CS:WORD PTR UPPER_CORNER,AX ;Save upper left corner

        MOV     SI,NUM_ROWS[BP]   ;Point to number of rows
        ADD     AH,[SI]           ;Add to start row to get last row

        MOV     SI,NUM_COLUMNS[BP] ;Point to number of columns
        ADD     AL,[SI]           ;Calculate last column number

        MOV     CS:WORD PTR LOWER_CORNER,AX ;Save lower right corner

        MOV     SI,RESPONSE[BP]   ;Point to response string
        CLD                     ;Increment index on string operation
        LODSW                     ;Get length of the string
        MOV     CS:WORD PTR RES_LEN,AX ;Put length in input buffer and

```

AGIOS Function Calls from High-Level Languages

```

        LODSW                      ;Get string starting address
        MOV     CS:WORD PTR [RES_STRING],AX ;Put string offset
                                           ; in input buffer
        MOV     AX,DS                ;Put string segment in input buffer
        MOV     CS:WORD PTR [RES_STRING+2],AX

        MOV     AX,4403H             ;IOCTL write (MSDOS AIOS call)
        MOV     BX,1                 ;console output handle
        MOV     CX,OFFSET LEN_AIOS_CAL ;Length of input buffer
        MOV     DX,OFFSET AIOS_CALL  ;Get offset of input buffer
        PUSH    CS                   ;Get segment of input buffer
        POP     DS                   ; into DS

        INT     21H                  ;Go define the touch field

        POP     DS                   ;Restore incoming DS
        POP     BP                   ;Restore incoming BP
        RET     STACK_SIZE           ;Clean up stack and return
page
AIOS_CALL:
        DW      32                   ;Define touch field function code

RES_STRING LABEL WORD
        DW      2                    DUP(?) ;Allocate two words for address
                                           ; of response string

RES_LEN LABEL WORD
        DW      ?                    ;Length of response string
        DB      2                    ;Report on release
        DB      1                    ;ASCII touch attribute
        DB      'J'                  ;Off enhancement - half bright inverse
        DB      'B'                  ;On enhancement - Full bright inverse
        DB      0                    ;Do not beep
        DB      0                    ;Do not position cursor

UPPER_CORNER LABEL WORD
        DW      ?                    ;Upper left coordinates

LOWER_CORNER LABEL WORD
        DW      ?                    ;Lower right coordinates

LEN_AIOS_CALL EQU $-AIOS_CALL ;Length of input buffer

FNTOUCH ENDP

```

AGIOS Function Calls from High-Level Languages

```

;*****
;
; OFFTOUCH
;
; Deletes all previously defined touch fields and sets the reporting
; mode to that specified by the called program.
;
; There is no functional return.
;*****
;

PUBLIC OFFTOUCH

OFFTOUCH PROC FAR

    PUSH DS ;Save the incoming DS register
    PUSH CS ;Set DS to current code segment
    POP DS

    MOV AX,4403H ;I/O control write
    MOV BX,1 ;Console device handle
    MOV CX,OFFSET LEN_TURN_IT_OFF ;Length of the buffer
    MOV DX,OFFSET TURN_IT_OFF ;Address of the buffer
    INT 21H ;Go delete fields

    MOV AX,4403H ;I/O control write
    MOV BX,1 ;Console device handle
    MOV CX,OFFSET LEN_REPORT_MODE ;Length of buffer
    MOV DX,OFFSET REPORT_MODE ;Select field-reporting mode
    INT 21H

    POP DS ;Restore incoming DS
    RET

TURN_IT_OFF:
    DW 34 ;Turn off touch function code
    DW OFFFFFH ;Delete all touch fields

LEN_TURN_IT_OFF EQU $-TURN_IT_OFF ;Length of the command

REPORT_MODE:
    DW 36 ;Select report mode
    DW 12 ;Report fields
    DW 2 ;Report on releases only

LEN_REPORT_MODE EQU $-REPORT_MODE

OFFTOUCH ENDP

CODE ENDS

END

```


AGIOS FUNCTION CALL EXAMPLES

APPENDIX

C

As mentioned in earlier sections, you can access AGIOS functions from high-level languages, such as BASIC or Pascal.

This appendix has examples on AGIOS video functions that are callable from Pascal. These video functions are Define Area, Write Area, Clear Area, Enhance Area, Read Area, Shift Area, and Write Line.

There is a comment block at the beginning of each example. The comment block describes each parameter in the parameter lists.

These examples get the input parameters from the stack and store them in an input buffer labeled AIOS_CMD. Once AIOS_CMD and other input registers are set up, AGIOS is called to execute the desired function. These example also assume that data resides in the default data segment pointed by the DS register.

Appendix B has detailed information on calling AGIOS functions from high-level languages.

NELSIS
THT - EL

AGIOS Function Call Examples

DEFINE AREA

```
*****
;
;
; FUNCTION DEF_AREA( UPPER_ROW:      INTEGER;
;                   LOWER_ROW:      INTEGER;
;                   LEFT_COLUMN:    INTEGER;
;                   RIGHT_COLUMN:   INTEGER):INTEGER;
;
;
; This function defines an area for the other video intrinsics.
;
; UPPER_ROW -      The upper row number of the area (top edge is 0)
; LOWER_ROW -      The bottom row number of the area (bottom edge is 47)
; LEFT_COLUMN -    The leftmost column of the area (left edge is 0)
; RIGHT_COLUMN -   The rightmost column of the area (bottom edge is 47)
;
; DEF_AREA -       Returns 0 if the call was successful
;
*****
```

AIOS_ENTRY	EQU	4403H	;I/O Control Write
AIOS_HANDLE	EQU	1	;Console handle
MSDOS	EQU	21H	;MSDOS entry point

AIOS_DATA	GROUP	AIOS_DATA
AIOS_CODE	GROUP	AIOS_CODE

AIOS_DATA	SEGMENT	BYTE PUBLIC 'AIOS_DATA'
	ASSUME	DS:AIOS_DATA

AIOS_CMD	DW	1	;Define Area function code
LR_COL	DB	?	;Lower right column
LR_ROW	DB	?	;Lower right row
UL_COL	DB	?	;Upper left column
UL_ROW	DB	?	;Upper left row
PREV_COORD	DW	-1,-1	;Don't return previous ;coordinates
AIOS_LEN	EQU	\$-AIOS_CMD	;Length of the input buffer
AIOS_DATA	ENDS		

AGIOS Function Call Examples

```

AIOS_CODE      SEGMENT BYTE PUBLIC 'AIOS_CODE'
                  ASSUME  CS:AIOS_CODE, DS:AIOS_DATA, ES:AIOS_DATA

UPPER_ROW      EQU      12                ;Stack offset for fetching
LOWER_ROW      EQU      10                ; parameters from the
LEFT_COLUMN    EQU      8                 ; stack
RIGHT_COLUMN    EQU      6
POP_PARMS      EQU      8                ;Total parameter size (in bytes)

DEF_AREA       PUBLIC DEF_AREA            ;DEF_AREA entry point
PROC           FAR

                PUSH     BP                ;Save incoming BP
                MOV      BP,SP             ;Set up BP to read parameter
                PUSH     DS                ;Save incoming DS

                MOV      AX,AIOS_DATA      ;Set ES & DS to specified data segment
                MOV      DS,AX
                MOV      ES,AX

                CLD                        ;Set flag to increment DI on string op
                MOV      DI,OFFSET LR_COL  ;Point to AIOS buffer data structure

                MOV      AL,RIGHT_COLUMN[BP] ;Get the right column number from stack
                STOSB                      ;Move right column in AIOS buffer
                MOV      AL,LOWER_ROW[BP]   ;Move lower row number in AIOS buffer
                STOSB                      ;Move left column into AIOS buffer
                MOV      AL,LEFT_COLUMN[BP] ;Move upper row number in AIOS buffer
                STOSB                      ;Move upper row number in AIOS buffer

                MOV      AX,AIOS_ENTRY      ;Set up registers to call AIOS
                MOV      BX,AIOS_HANDLE
                MOV      CX,AIOS_LEN
                MOV      DX,OFFSET AIOS_CMD
                INT      MSDOS              ;Go define the area - recall that
                                           ; AX=0 implies call was succesful
                                           ;Restore incoming registers

                POP      DS
                POP      BP
                RET      [POP_PARMS]        ;Toss parameters and return

DEF_AREA       ENDP
AIOS_CODE      ENDS

END

```

AGIOS Function Call Examples

WRITE AREA

```

*****
;
;
;      FUNCTION  WRITE_AREA(  ARRAY_LEN:      INTEGER;
;                          _VAR  CHARACTERS:    CHAR;
;                          VAR   CHAR_SETS:     CHAR;
;                          VAR   ENHANCEMENTS:  CHAR):INTEGER;
;
;
;      This function writes characters, enhancements, and or
;      character sets to the previously defined area.
;
;      ARRAY_LEN -      The length of the array(s)
;      CHARACTERS -      Offset pointing to the first character in a
;                          stream of characters to be written to the area
;      ENHANCEMENTS -      Offset of the first enhancement code
;      CHAR_SETS -      Offset of the first character set code
;
;      WRITE_AREA -      Returns 0 if the call was successful
;
;      If the first character is OFFH, then this routine assumes that
;      there aren't any data associated with that parameter.  The
;      segment will be set to -1 for AIOS to ignore that parameter.
;
;
*****

AIOS_ENTRY      EQU      4403H          ;I/O control write
AIOS_HANDLE      EQU      1             ;Console device handle
MSDOS           EQU      21H           ;MSDOS entry point

AIOS_DATA        GROUP    AIOS_DATA
AIOS_CODE        GROUP    AIOS_CODE

AIOS_DATA        SEGMENT BYTE PUBLIC 'AIOS_DATA'
                  ASSUME  DS:AIOS_DATA

AIOS_CMD         DW        2            ;Write Area function code
DATA_LENGTH      DW        ?            ;Length of the data arrays
ENH_PTR          DD        ?            ;Points to enhancement codes
CHAR_SET_PTR     DD        ?            ;Points character set codes
ASCII_PTR        DD        ?            ;Points to character codes

```

AGIOS Function Call Examples

```

AIOS_LEN      EQU      $-AIOS_CMD      ;Length of the AIOS buffer
AIOS_DATA     ENDS

AIOS_CODE     SEGMENT BYTE PUBLIC 'AIOS_CODE'
ASSUME CS:AIOS_CODE, DS:NOTHING, ES:AIOS_DATA

ARRAY_LEN     EQU      12              ;Locate the input parameters
CHARACTERS    EQU      10              ; passed on the stack
CHAR_SETS     EQU      8
ENHANCEMENTS  EQU      6
POP_PARMS     EQU      8              ;Total parameter size (in bytes)

WRITE_AREA    PUBLIC WRITE_AREA        ;WRITE_AREA entry point
PROC          FAR

      PUSH     BP                      ;Save Incoming BP
      MOV      BP,SP                  ;Set up BP to read parameters
      PUSH     DS                      ;Save incoming DS

      MOV      BX,AIOS_DATA           ;Set ES to specified data segment
      MOV      ES,BX

      CLD                             ;Set flag to increment on string op
      MOV      DI,OFFSET ENH_PTR      ;Point to AIOS buffer structure

      MOV      SI,ENHANCEMENTS[BP]    ;Set up enhancement pointer
      CALL     LOAD_PARM
      MOV      SI,CHAR_SETS[BP]       ;Set up character sets pointer
      CALL     LOAD_PARM
      MOV      SI,CHARACTERS[BP]      ;Set up characters pointer
      CALL     LOAD_PARM

      ASSUME   DS:AIOS_DATA           ;Set DS to defined data segment
      MOV      DS,BX

      MOV      AX,ARRAY_LEN[BP]        ;Move length into AIOS buffer
      MOV      DATA_LENGTH,AX

      MOV      AX,AIOS_ENTRY          ;Set up parameters for AIOS
      MOV      BX,AIOS_HANDLE
      MOV      CX,AIOS_LEN
      MOV      DX,OFFSET AIOS_CMD
      INT      MSDOS                  ;Write area and recall that AX=0
                                      ; implies the call was successful
      POP      DS                     ;Restore incoming registers
      POP      BP
      RET     [POP_PARMS]              ;Toss parameters and return

WRITE_AREA    ENDP

```

AGIOS Function Call Examples

```

;
;The following routine moves the pointer from (DS:SI) to (ES:DI)
;
LOAD_PARM    PROC     NEAR
    CMP      BYTE PTR [SI],OFFH      ;Is first byte = OFFH?
    JZ       NULL_PTR               ;YES - put null into AIOS buffer
    MOV      ES:[DI],SI              ;NO - put the pointer (DS:SI)
    MOV      ES:[DI+2],DS            ;      into (ES:DI)
    ADD      DI,4                    ;Adjust ES:DI for next pointer
    RET

NULL_PTR:
    MOV      AX,OFFFFH               ;OFFFFH indicates a null pointer
    STOSW                                ;Put a null pointer into
    STOSW                                ; AIOS buffer and add advance
    RET                                ; index

LOAD_PARM    ENDP

AIOS_CODE    ENDS

END

```

CLEAR AREA

```

;*****
;
;      FUNCTION  CLEAR_AREA      :INTEGER;
;
;      This function clears the entire defined area (characters sets
;      to blanks, enhancements to 'none', and characters to 'Roman').
;
;      CLEAR_AREA -      Returns 0 if the call was successful
;
;*****

```

```

AIOS_ENTRY      EQU      4403H      ;I/O control write
AIOS_HANDLE     EQU      1          ;Console device handle
MSDOS           EQU      21H        ;MSDOS entry point

AIOS_DATA       GROUP    AIOS_DATA
AIOS__CODE      GROUP    AIOS_CODE

AIOS_DATA       SEGMENT BYTE PUBLIC 'AIOS_DATA'
                ASSUME   DS:AIOS_DATA

AIOS_CMD        DW      3           ;Clear Area function code

AIOS_LEN        EQU      $-AIOS_CMD ;Length of the AIOS buffer

AIOS_DATA       ENDS

AIOS_CODE       SEGMENT BYTE PUBLIC 'AIOS_CODE'
                ASSUME   CS:AIOS_CODE, DS:AIOS_DATA

POP_PARAMS      EQU      0          ;Total parameter size (in bytes)

```

AGIOS Function Call Examples

```
CLEAR_AREA      PUBLIC  CLEAR_AREA      ;CLEAR_AREA entry point
PROC            FAR
    PUSH        DS                      ;Save incoming DS
    MOV         AX,AIOS_DATA            ;Set DS to specified data area
    MOV         DS,AX
    MOV         AX,AIOS_ENTRY           ;Set up parameters for the
    MOV         BX,AIOS_HANDLE          ; call
    MOV         CX,AIOS_LEN
    MOV         DX,OFFSET AIOS_CMD
    INT         MSDOS                  ;Go clear the area
                                           ;AX = 0 implies no error
    POP         DS                      ;Restore incoming registers
    RET                               ;Toss parameters and return

CLEAR_AREA      ENDP
AIOS_CODE       ENDS

END
```

ENHANCE AREA

```

;*****
;
;      FUNCTION  ENH_AREA( ENHANCEMENT: CHAR):INTEGER;
;
;      This function ehnhances the entire defined area with
;      the selected ENHANCEMENT
;
;      ENHANCEMENT -   The desired enhancement type (a letter from
;                      'a' .. 'z')
;
;      ENH_AREA -      Returns 0 if the call was successful
;
;*****

```

```

AIOS_ENTRY      EQU      4403H           ;I/O control write
AIOS_HANDLE     EQU      1              ;Console device handle
MSDOS           EQU      21H           ;MSDOS entry point

AIOS_DATA       GROUP    AIOS_DATA
AIOS_CODE       GROUP    AIOS_CODE

AIOS_DATA       SEGMENT BYTE PUBLIC 'AIOS_DATA'
                ASSUME    DS:AIOS_DATA

AIOS_CMD        DW       4              ;Enhance Area function code
ENHANCE         DW       ?              ;Allocate a word for enhancement

AIOS_LEN        EQU      $-AIOS_CMD     ;Length of the AIOS buffer

AIOS_DATA       ENDS

AIOS_CODE       SEGMENT BYTE PUBLIC 'AIOS_CODE'
                ASSUME    CS:AIOS_CODE, DS:AIOS_DATA

ENHANCEMENT     EQU      6              ;Offset to fetch parameters
POP_PARMS       EQU      2              ;Total parameters size (in bytes)

```

AGIOS Function Call Examples

ENH_AREA	PUBLIC PROC	ENH_AREA FAR	;ENH_AREA entry point
PUSH	BP		;Save Incoming BP
MOV	BP,SP		;Set up BP to read param
PUSH	DS		;Save incoming DS
MOV	AX,AIOS_DATA		;Set DS to specified data area
MOV	DS,AX		
MOV	AL,ENHANCEMENT[BP]		;Get specified enhancement code
MOV	BYTE PTR ENHANCE,AL		;Put enhancement in AIOS buffer
MOV	AX,AIOS_ENTRY		;Set up parameters for the
MOV	BX,AIOS_HANDLE		; call
MOV	CX,AIOS_LEN		
MOV	DX,OFFSET AIOS_CMD		
INT	MSDOS		;Go enhance the area
			;AX = 0 implies the function
			; was successful
POP	DS		;Restore incoming registers
POP	BP		
RET	[POP_PARMS]		;Toss parameters and return
ENH_AREA	ENDP		
AIOS_CODE	ENDS		
END			

READ AREA

```

;*****
;
;      FUNCTION  READ_AREA( ARRAY_LEN:      INTEGER;
;                          VAR  CHARACTERS:  CHAR;
;                          VAR  CHAR_SETS: CHAR;
;                          VAR  ENHANCEMENTS: CHAR):INTEGER;
;
;      This function reads characters, enhancements, and or
;      character sets from the previously defined area of the screen.
;
;      ARRAY_LEN -      The length of the array(s)
;      CHARACTERS -      Offset pointing to the first location for a
;                          character to be returned from the area
;      ENHANCEMENTS -      Offset of the first enhancement code
;      CHAR_SETS -      Offset of the first character set code
;
;      READ_AREA -      Returns 0 if the call was successful
;
;      If the first character is OFFH, then this routine assumes that
;      there aren't any data associated with that parameter. The
;      segment will be set to -1 to tell AIOS to ignore that parameter.
;
;*****

```

```

AIOS_ENTRY      EQU      4403H          ;I/O control write
AIOS_HANDLE      EQU      1             ;Console device handle
MSDOS            EQU      21H           ;MSDOS entry point

AIOS_DATA        GROUP    AIOS_DATA
AIOS__CODE        GROUP    AIOS_CODE

AIOS_DATA        SEGMENT BYTE PUBLIC 'AIOS_DATA'
ASSUME DS:AIOS_DATA

AIOS_CMD          DW      5             ;Read Area function code
DATA_LENGTH       DW      ?            ;Length of the data arrays
ENH_PTR           DD      ?            ;Points to enhancement codes
CHAR_SET_PTR      DD      ?            ;Points character set codes
ASCII_PTR         DD      ?            ;Points to characters

AIOS_LEN          EQU      $-AIOS_CMD   ;Length of the AIOS buffer
AIOS_DATA         ENDS

AIOS_CODE         SEGMENT BYTE PUBLIC 'AIOS_CODE'
ASSUME CS:AIOS_CODE, DS:NOTHING, ES:AIOS_DATA

```

AGIOS Function Call Examples

ARRAY_LEN	EQU	12	;Offset to fetch parameters
CHARACTERS	EQU	10	; on the stack
CHAR_SETS	EQU	8	
ENHANCEMENTS	EQU	6	
POP_PARMS	EQU	8	;Total parameter size (in bytes)
	PUBLIC	READ_AREA	;READ_AREA entry point
READ_AREA	PROC	FAR	
PUSH	BP		;Save Incoming BP
MOV	BP,SP		;Set up BP to read parameters
PUSH	DS		;Save incoming DS
MOV	BX,AIOS_DATA		;Set ES to specified data area
MOV	ES,BX		
CLD			;Set flag to increment index
MOV	DI,OFFSET ENH_PTR		;Point to specified data structure
MOV	SI,ENHANCEMENTS[BP]		;Set up enhancement pointer
CALL	LOAD_PARM		
MOV	SI,CHAR_SETS[BP]		;Set up character sets pointer
CALL	LOAD_PARM		
MOV	SI,CHARACTERS[BP]		;Set up characters pointer
CALL	LOAD_PARM		
ASSUME	DS:AIOS_DATA		;Set DS to specified data structure
MOV	DS,BX		;DS still has AIOS_DATA
MOV	AX,ARRAY_LEN[BP]		;Move length into AIOS buffer
MOV	DATA_LENGTH,AX		
MOV	AX,AIOS_ENTRY		;Set up parameters for the
MOV	BX,AIOS_HANDLE		; call
MOV	CX,AIOS_LEN		
MOV	DX,OFFSET AIOS_CMD		
INT	MSDOS		;Read screen data & return
			; completion code
POP	DS		;Restore incoming registers
POP	BP		
RET	[POP_PARMS]		;Toss parameters and return
READ_AREA	ENDP		

AGIOS Function Call Examples

```

;
;The following routine moves the pointer from (DS:SI) to (ES:DI)
;
LOAD_PARM      PROC    NEAR      ;This proc sets up the AIOS pointer
                                ;DS:SI) It is placed in ES:DI
                                ;Is 1st byte OFFH (A null pointer
                                ;YES - put null into AIOS buffer
                                ;No, put the pointer (DS:SI)
                                ; in the AIOS command
                                ;Adjust ES:DI for next pointer
                                CMP    BYTE PTR [SI],OFFH
                                JZ     NULL_PTR
                                MOV    ES:[DI],SI
                                MOV    ES:[DI+2],DS
                                ADD    DI,4
                                RET

NULL_PTR:
    MOV    AX,OFFFFH      ;OFFFFH indicates a null
    STOSW      ;Put a null pointer into
    STOSW      ;AIOS buffer and advance index
    RET

LOAD_PARM      ENDP

AIOS_CODE      ENDS

END

```

SHIFT AREA

```
*****
;
; FUNCTION SHIFT_AREA( ARRAY_LEN:      INTEGER;
;                       DIRECTION:      INTEGER;
;                       DISTANCE:       INTEGER;
;                       VAR CHARACTERS:  CHAR;
;                       VAR CHAR_SETS:   CHAR;
;                       VAR ENHANCEMENTS: CHAR):INTEGER;
;
; This function shifts the defined area (if characters, character
; sets, or enhancements are supplied, they will be assigned to the
; vacated area).
;
; ARRAY_LEN - The length of the array(s)
; DIRECTION - Shift direction; 0: up, 1: down, 2: left, 3: right
; DISTANCE - Number of rows or columns to shift
; CHARACTERS - Character pointer
; ENHANCEMENTS - Enhancement pointer
; CHAR_SETS - Character Set pointer
;
; SHIFT_AREA - Returns 0 if the call was successful
;
; When the first character of CHARACTERS, CHAR_SETS, or ENHANCEMENTS
; is OFFH, the AIOS pointer will be set null (segment is OFFFFH).
;
*****

AIOS_ENTRY EQU 4403H ;I/O control write
AIOS_HANDLE EQU 1 ;Console device handle
MSDOS EQU 21H ;MSDOS entry point

AIOS_DATA GROUP AIOS_DATA
AIOS_CODE GROUP AIOS_CODE

AIOS_DATA SEGMENT BYTE PUBLIC 'AIOS_DATA'
ASSUME DS:AIOS_DATA
```

AGIOS Function Call Examples

```

AIOS_CMD      DW      6                ;Shift Area function code
DATA_LENGTH   DW      ?                ;Length of the data arrays
ENH_PTR       DD      ?                ;Points to enhancements
CHAR_SET_PTR  DD      ?                ;Points the character sets
ASCII_PTR     DD      ?                ;Points to characters
SHIFT_DIST    DB      ?                ;Number of rows or columns to shift
SHIFT_DIR     DB      ?                ;Direction to shift
AIOS_LEN      EQU     $-AIOS_CMD       ;Length of the AIOS buffer
AIOS_DATA     ENDS

```

```

AIOS_CODE     SEGMENT BYTE PUBLIC 'AIOS_CODE'
ASSUME CS:AIOS_CODE, DS:NOTHING, ES:AIOS_DATA

```

```

ARRAY_LEN     EQU     16                ;Offset to fetch parameters
DIRECTION     EQU     14
DISTANCE      EQU     12
CHARACTERS    EQU     10
CHAR_SETS     EQU     8
ENHANCEMENTS  EQU     6
POP_PARMS     EQU     12                ;Total parameter size (in byte)
PUBLIC SHIFT_AREA
SHIFT_AREA    PROC     FAR                ;SHIFT_AREA entry point

    PUSH      BP                        ;Save Incoming BP
    MOV       BP,SP                     ;Set up BP to read parameters
    PUSH      DS                        ;Save incoming DS

    MOV       BX,AIOS_DATA              ;Set ES to specified data area
    MOV       ES,BX

    CLD
    MOV       DI,OFFSET DATA_LENGTH    ;Set flag to increment index
                                           ;Point to specified data structure

    MOV       AX,ARRAY_LEN[BP]           ;Get array length from stack
    MOV       SI,ENHANCEMENTS[BP]       ;Move array length into AIOS buffer
    CALL      LOAD_PARM                  ;Set up enhancement pointer

    MOV       SI,CHAR_SETS[BP]           ;Set up character sets pointer
    CALL      LOAD_PARM
    MOV       SI,CHARACTERS[BP]          ;Set up characters pointer
    CALL      LOAD_PARM

    MOV       AL,DISTANCE[BP]            ;Move distance parameter into
    STOSB                                   ;   AIOS buffer
    MOV       AL,DIRECTION[BP]           ;Move the direction parameter
    STOSB                                   ;   into the AIOS buffer

```

AGIOS Function Call Examples

```

MOV     DS,BX                ;Set DS to specified data segment
                                ;DS still has AIOS_DATA
MOV     AX,AIOS_ENTRY        ;Set up parameters for the call
MOV     BX,AIOS_HANDLE
MOV     CX,AIOS_LEN
MOV     DX,OFFSET AIOS_CMD
INT     MSDOS                ;Shift the area & set return
                                ; status
POP     DS                    ;Restore incoming registers
POP     BP
RET     [POP_PARMS]          ;Toss parameters and return
SHIFT_AREA ENDP
;
;The following procedure moves the pointer from (DS:SI) to (ES:DI).
;
LOAD_PARM PROC NEAR        ;This procedure sets up the data pointer

CMP     BYTE PTR [SI],OFFH   ;Is first byte OFFH?
JZ      NULL_PTR            ;YES - put null into AIOS buffer
MOV     ES:[DI],SI          ;NO - put the pointer (DS:SI)
MOV     ES:[DI+2],DS        ; in the AIOS buffer
ADD     DI,4                ;Adjust ES:DI for next pointer
RET

NULL_PTR:
MOV     AX,OFFFFH           ;OFFFFH indicates a null
STOSW                                     ;Put a null pointer into
STOSW                                     ; AIOS buffer and advance index
RET

LOAD_PARM ENDP
AIOS_CODE ENDS
END

```

WRITE LINE

```

*****
;
;      FUNCTION  WRITE_LINE(  ARRAY_LEN:      INTEGER;
;                           COLUMN:          INTEGER;
;                           ROW:             INTEGER;
;                           VAR  CHARACTERS:   CHAR;
;                           VAR  CHAR_SETS:    CHAR;
;                           VAR  ENHANCEMENTS: CHAR):INTEGER;
;
;      This function writes characters, enhancements, and or
;      character sets to a specified line on the screen.
;
;      ARRAY_LEN -   The length of the array(s)
;      ROW -        The row number where the write will begin
;                  (Remember that rows & columns are 0 based.)
;      COLUMN -     The column number where the write will begin
;      CHARACTERS - Pointer to the first character
;      ENHANCEMENTS - Pointer to the first enhancement code
;      CHAR_SETS -  Pointer to the first character set code
;
;      WRITE_LINE - Returns 0 if the call was successful
;
;      If the first byte of CHARACTERS, ENHANCEMENTS, or CHAR_SETS is
;      OFFH, then a null pointer will be passed to AIOS.
;
*****

AIOS_ENTRY    EQU    4403H           ;I/O control write
AIOS_HANDLE   EQU    1              ;Console device handle
MSDOS         EQU    21H            ;MSDOS entry point

AIOS_DATA     GROUP  AIOS_DATA
AIOS_CODE     GROUP  AIOS_CODE
AIOS_DATA     SEGMENT BYTE PUBLIC 'AIOS_DATA'
               ASSUME DS:AIOS_DATA

```

AGIOS Function Call Examples

AIOS_CMD	DW	7	;Write Line function code
COL_NUMBER	DB	?	;Column number to start
ROW_NUMBER	DB	?	;Row number to start
DATA_LENGTH	DW	?	;Length of the data arrays
ENH_PTR	DD	?	;Points to enhancements
CHAR_SET_PTR	DD	?	;Points to character sets
ASCII_PTR	DD	?	;Points to characters
AIOS_LEN	EQU	\$-AIOS_CMD	;Length of the AIOS buffer
AIOS_DATA	ENDS		
AIOS_CODE	SEGMENT BYTE PUBLIC 'AIOS_CODE'		
	ASSUME CS:AIOS_CODE, DS:NOTHING, ES:AIOS_DATA		
ARRAY_LEN	EQU	16	;Offset to fetch input parameters
ROW	EQU	14	
COLUMN	EQU	12	
CHARACTERS	EQU	10	
CHAR_SETS	EQU	8	
ENHANCEMENTS	EQU	6	
POP_PARMS	EQU	12	;Total parameter size (in bytes)
	PUBLIC	WRITE_LINE	;WRITE_LINE entry point
WRITE_LINE	PROC	FAR	
PUSH	BP		;Save Incoming BP
MOV	BP,SP		;Set up BP to read parameters
PUSH	DS		;Save incoming DS
MOV	EX,AIOS_DATA		;Set ES to specified data area
MOV	ES,BX		
CLD			;Set flag to increment index
MOV	DI,OFFSET COL_NUMBER		;Point to specified data structure
MOV	AL,COLUMN[BP]		;Move row number into the
STOSB			; AIOS buffer
MOV	AL,ROW[BP]		;Move column number into AIOS buffer
STOSB			
MOV	AX,ARRAY_LEN[BP]		;Move data length into AIOS buffer
STOSW			
MOV	SI,ENHANCEMENTS[BP]		;Set up enhancement pointer
CALL	LOAD_PARM		
MOV	SI,CHAR_SETS[BP]		;Set up character set pointer
CALL	LOAD_PARM		
MOV	SI,CHARACTERS[BP]		;Set up character pointer
CALL	LOAD_PARM		

AGIOS Function Call Examples

```

ASSUME DS:AIOS_DATA      ;Set DS to specified data area
MOV     DS,EX             ;DS still has AIOS_DATA

MOV     AX,AIOS_ENTRY     ;Set up parameters for the call
MOV     BX,AIOS_HANDLE
MOV     CX,AIOS_LEN
MOV     DX,OFFSET AIOS_CMD
INT     MSDOS             ;Write line and return completion
                                ; status
POP     DS               ;Restore incoming registers
POP     BP
RET     [POP_PARMS]       ;Toss parameters and return

WRITE_LINE      ENDP

;
;The following procedure moves the pointer from (DS:SI) to (ES:DI).
;

LOAD_PARM      PROC      NEAR      ;This routine moves a pointer
                                ;into the AIOS buffer
                                ;Is first byte OFFH?
CMP     BYTE PTR [SI],OFFH
JZ      NULL_PTR          ;YES - put null into AIOS buffer
MOV     ES:[DI],SI        ;NO - put the pointer (DS:SI)
MOV     ES:[DI+2],DS      ; into the AIOS buffer
ADD     DI,4              ;Adjust ES:DI for next pointer
RET

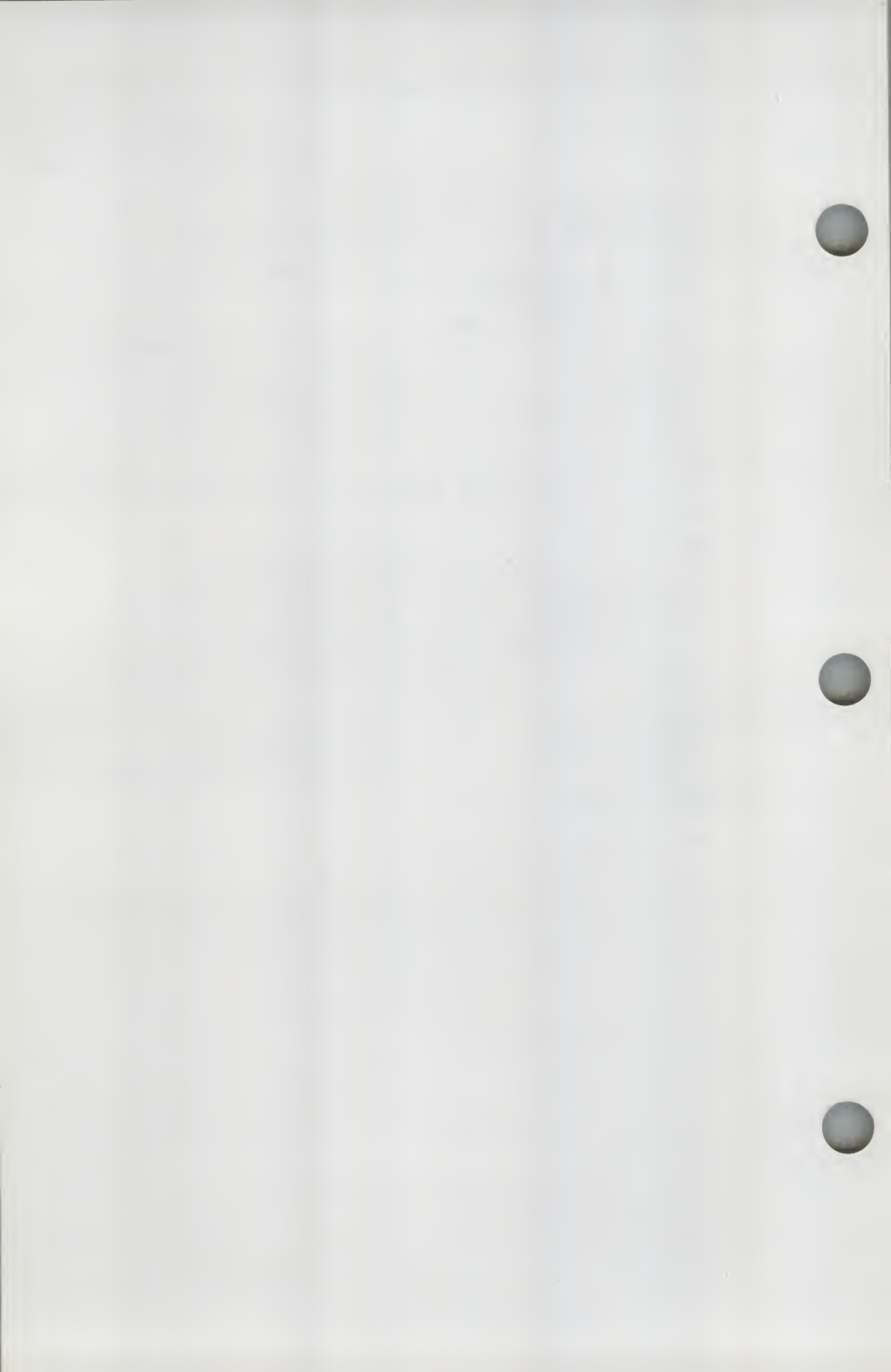
NULL_PTR:
MOV     AX,OFFFHH         ;OFFFHH indicates a null
STOSW   ;Put a null pointer into
STOSW   ; AIOS command and advance index
RET

LOAD_PARM      ENDP

AIOS_CODE      ENDS

END

```



APPENDIX D

GRAPHICS CONTROL FUNCTIONS

A logo consisting of a rounded rectangle with a thick black border. Inside the rectangle, the word "NELSIS" is written in a large, bold, sans-serif font. Below it, "THT - EL" is written in a smaller, bold, sans-serif font.

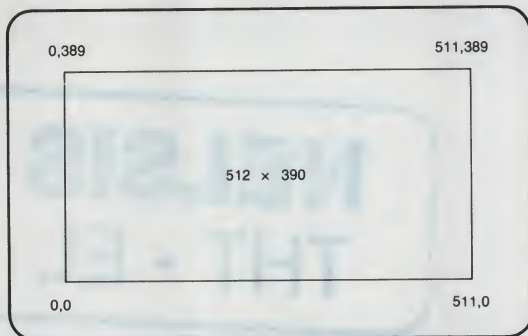
NELSIS
THT - EL

Introduction

This section contains a description of the terminal's graphic functions and how they are used. The information and the examples are intended for use in developing programs to control the graphics functions. Additional information on how to use the graphics features from the keyboard is contained in the **HP150 Terminal User's Guide**.

Graphics Display

You can display graphics data by addressing points in a 512 by 390 array.



The graphics and alphanumeric data are displayed in the same area on the screen but are stored in separate RAM memories. This allows you to read or modify graphics and alphanumeric data separately.

NOTE

Alphanumeric characters overlay or mask out the graphics display. Thus enhancements (blinking, inverse video, etc.) alter the appearance of graphics already on the screen. The data in the graphics display memory, however, remains unchanged.

Keyboard Graphics Functions

Graphics functions commands in the form of escape sequences can be entered at the terminal keyboard by the operator. In addition, certain graphics functions are performed via the set of graphics control keys located to the right of the normal ASCII character set (see figure D-1). Table D-1 contains a list of the keys and a description of their functions. These keys function in either local or remote operation. This allows a combination of operator and program control of graphics functions. Refer to the **HP150 Terminal User's Guide** for detailed information on using the graphics control keys.

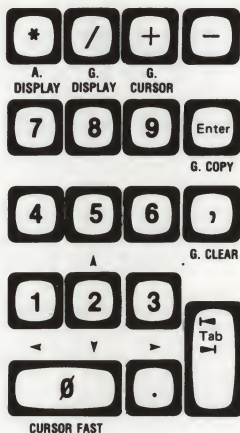


Figure D-1. Location of Graphics Keys

Each key (except the TAB key, the decimal point key, and the 4, 6, 7, 8, and 9 keys) performs one of two functions, depending upon the current state of the keypad. Pressing **CTRL**, and the **⏏** key on the graphics keypad toggles the keypad between graphics and numerics functions. The current state of the keypad appears as **Grph Pad** or **Num Pad** in the status line at the bottom of the screen display (the 27th row). The default state is **Num Pad**. The current state of the keypad, however, is stored in non-volatile memory; if you set the keypad to **Grph Pad** and turn off the terminal, the keypad is still in **GRAPH PAD** state when you reapply power. Some applications will change the state of the keypad as needed. Hard reset (**SHIFT** **CTRL** **RESET**) does not affect the keypad state.

You can set the keypad programmatically using the command:

EC & k <parameter> o




where <parameter> may be:

0 = Num Pad

1 = Grph Pad

While in Grph Pad, the cursor control keys are the only repeating keys. Also, in Grph Pad the minus key on the **Num Pad** and the **TAB**, **.**, and **4**, **6**, **7**, **8**, and **9** keys are disabled.

Table D-1. Graphics Control Keys

KEY	DESCRIPTION
Graph cursor	Toggles the graphics cursor on and off.
	Move the graphics cursor. More than one key can be pressed for diagonal motion.
Cursor fast	Speeds up the graphics cursor if pressed in conjunction with the cursor keys. The rate returns to normal when released.
Graph display	Toggles the graphics display to inhibit the graphics image without erasing.
Alpha display	Toggles the alphanumeric display to inhibit the alphanumeric image without erasing.
Graph clear	Erases the graphics image memory.
• Graph copy	Copies graphics memory to the specified "to" devices.
	Toggles the function of the keypad between graphics and numerics when CTRL is pressed simultaneously.
	Unshifted in numeric mode, the key is used to display a minus (-) character.

*The Graph copy key causes an error message to appear if no valid destination device is specified.

Syntax Notations for Graphics Escape Sequences

All terminal graphics functions can be accomplished through escape sequences. An escape sequence is a series of ASCII characters preceded by the escape character. The escape character signals the terminal that the succeeding characters form a command.

The two general forms of the graphics escape sequences are:

Ec * <FG> <FL> or <FU> ,

Ec * <FG> | <decimal parameter> | <FL> or <FU>

<FG> is the Function Group or family indicator which must be a lower case character from the set (a,b,c . . . z). For example:

Ec*d sequences are display commands

Ec*p sequences are plotting commands.

| | encloses one or more decimal parameters separated by spaces or commas. For example:

Ec*p <cx>,<cy>Z is a plotting command

Ec*pb 194, 250Z is the command to draw a vector from the current pen position to 194,250.

<> are angle brackets that always enclose character parameters. For example:

Ec*m <x>,<y>j sets the relocatable origin in absolute coordinates

Ec*m 255,294J sets the absolute relocatable origin at 244,194.

<FL> is a Lower case Function terminator from the set (a,b,c . . . z). For example:

Ec*d <z>k turns on the graphics cursor.

If <FL> is used to terminate a function, other function terminators from the same family or function group may be appended to the sequence until an upper case terminator <FU> is used.

FU> is an Upper case Function terminator from the set (A,B,C . . . Z, and ^). If <FU> is used, it terminates both the function and the escape sequence.

An escape character will also terminate this function and the escape sequence. For example:

Ec*p<cx>,<cy>Z - "Z" ends the sequence
Ec*m<x>Q - "Q" ends the sequence.

NOTE

Ec*1 (label) commands are terminated by carriage return, linefeed, or the escape character.

Miscellaneous Characters

No spaces are allowed among the first three characters of the escape sequence (for example Ec*d) but spaces after these are ignored (except in character parameters). Carriage returns and line feeds are ignored after the first three characters (except in the label commands — Ec*1).

Table D-2. Summary of Graphics Sequence Types

Escape Sequence	Description
Ec * d	Display Control
Ec * l	Labeling
Ec * m	Drawing Mode
Ec * p	Vector Plotting
Ec * s	Graphics Status
Ec * t	Compatibility Mode
Ec * w	Graphics Initialization

Control Codes

Control codes are generally ignored, with the exception of the ESCAPE character (Ec). If an Ec character is detected and the previous graphics control sequence has not been properly terminated with a "Z" or some other valid upper case character, the Ec will abort execution of the previous sequence and the new escape sequence will be executed.

Graphics Display Control

Graphics display control is made up of the functions used to control the graphics cursor, the portion of the graphics memory that is currently being displayed, or the state of the graphics memory. These functions are as follows:

- Graphics Cursor Control
- Graphics Memory Control

Table D-3 lists the escape sequences for the graphics display control functions.

Table D-3. Graphics Display Control Functions

FUNCTION	CODE	DESCRIPTION
Graphics Cursor Control		
Cursor On	Ec•dk	Turn on the graphics cursor.
Cursor Off	Ec•dl	Turn off the graphics cursor.
Move Absolute	Ec•d<x>,<y>o	Position the graphics cursor.
Move Relative	Ec•d<x>,<y>p	Position the graphics cursor.
Graphics Memory Control		
Clear Memory	Ec•da	Turn off all dots in graphics memory.
Set Memory	Ec•db	Turn on all dots in graphics memory.
Display On	Ec•dc	Enable the graphics display.
Display Off	Ec•dd	Inhibit the graphics display.

See the end of this chapter for a complete list of Graphics escape sequences.

Graphics Cursor Control

A separate graphics cursor is available for use in locating points in the graphics display. The graphics cursor is used by the terminal operator to input position data or to interact with a graphics application program.

GRAPHICS CURSOR ON/OFF. The graphics cursor is initially off (power on or full reset). Turning the cursor on or off does not affect the data in graphics memory.

The graphics cursor may be toggled on and off by pressing the **Graph cursor** key on the graphics/numeric pad.

Programmatically, you can toggle the cursor:

Graphics Cursor On: `Ec*dk`

Graphics Cursor Off: `Ec*d1`

GRAPHICS CURSOR POSITIONING. The graphics cursor is initially at position (0,0) after power on, full reset, graphics hard reset (`Ec*wr`), or graphics default reset (`Ec*mr`). You can position the cursor (even if it is not turned on) using either absolute or relative coordinates. In the following sequences, X and Y give the new cursor position. Refer to Vectors for a discussion of absolute and relative coordinates.

Position Graphics

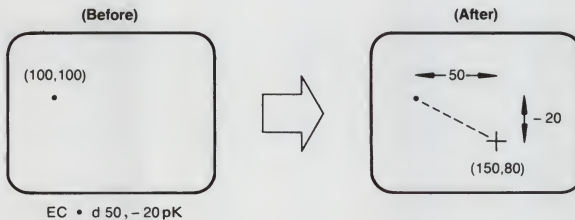
Cursor Absolute: `Ec*d<X>,<Y>o`

Position Graphics

Cursor Relative: `Ec*d<X>,<Y>p`

You can position the graphics cursor with the graphics/numeric keypad by pressing \div , \pm , \uparrow , \downarrow . Pressing two keys simultaneously causes diagonal movement. The **Cursor fast** key can be pressed at the same time for high-speed cursor positioning.

Example: The cursor is currently at position 100,100 and off. Move it 50 units to the right and 20 units down from its current position and turn it on.



Graphics Memory Control

The graphics display can be turned on or off or the entire memory can be set to all ones (dots on) or all zeros (dots off).

GRAPHICS DISPLAY ON/OFF. The graphics display and graphics cursor can be turned on or off. The data in the graphics memory is unaffected.

From the graphics/numeric keypad, pressing **Graph display** toggles the graphics display on and off:

Programmatically:

Graphics Display On: **Ec*dc**

Graphics Display Off: **Ec*dd**

GRAPHICS DISPLAY SET/CLEAR. The graphics data currently displayed on the screen can be set to all ones (a white screen) or cleared to all zeros (a black screen).

Clear Graphics Memory: **Ec*da**

Set Graphics Memory: **Ec*db**

You can clear graphics data on the screen from the keyboard by pressing **Graph clear** on the graphics/numeric pad.

Graphics Drawing Mode Parameters

There are several drawing parameters that can be set to allow a wide variety of drawing capabilities. These parameters select whether points will be turned on or turned off, define line or area patterns, position the relocatable origin, and define graphics text settings.

Graphics drawing mode control sequences begin with `Ec * m` followed by one or more of the drawing parameters. Table D-4 lists the mode control commands.

Table D-4. Graphics Mode Commands

<code>Ec * m <parameters></code>	
PARAMETERS	DESCRIPTION
<code>a</code>	select drawing mode
<code>b</code>	select line type
<code>c</code>	define user line pattern
<code>d</code>	define user area fill pattern
<code>e</code>	rectangular area fill, absolute
<code>f</code>	rectangular area fill, relocatable
<code>g</code>	select area fill pattern
<code>h</code>	select area boundary pen
<code>j</code>	set relocatable origin to absolute coordinates
<code>k</code>	set relocatable origin to current pen position
<code>l</code>	set relocatable origin to graphics cursor position
<code>m</code>	set graphics text size
<code>n</code>	set graphics text direction
<code>o</code>	turn on character slant
<code>p</code>	turn off character slant
<code>q</code>	set text origin
<code>r</code>	set graphics defaults
<code>z</code>	NOP

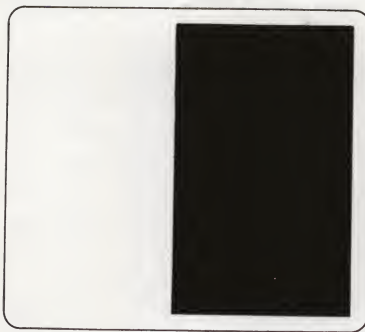
Drawing Modes

Vectors can be drawn by setting, clearing, or complementing the data in the graphics memory. Normally the memory is cleared and vectors are drawn by setting selected bits to make green lines on a dark screen. If instead you want black vectors on a green screen, you can begin by setting memory (refer to the section on Graphics Memory Control above), select a clear or complement line type and draw dark vectors (refer to the example that follows). Figure D-2 illustrates the various drawing modes.

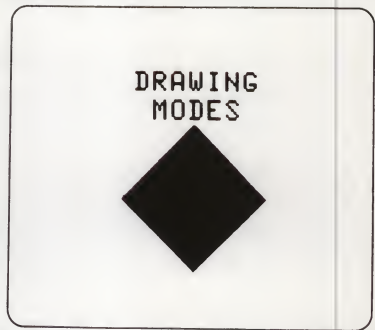
Set Drawing Mode: `Ec * m <parameter> a`

where: `<parameter>` is

- 0 = no effect
- 1 = Clear (turn off graphics bits)
- 2 = Set (turn on graphics bits)
- 3 = Complement (toggle the graphics bits)
- 4 = Jam (turn bits on or off according to the data)



Original Screen



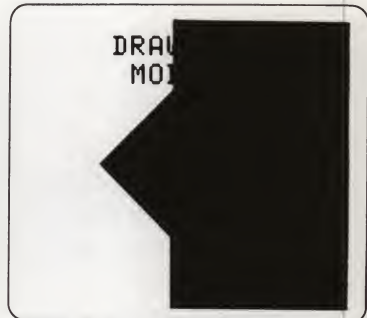
Pattern to be drawn over Original Screen



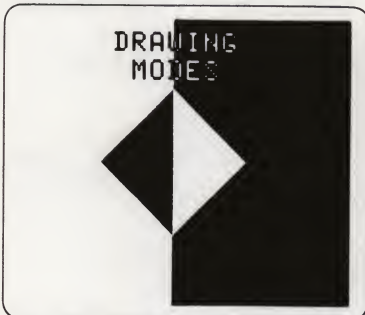
Mode 0 (No effect on original screen)



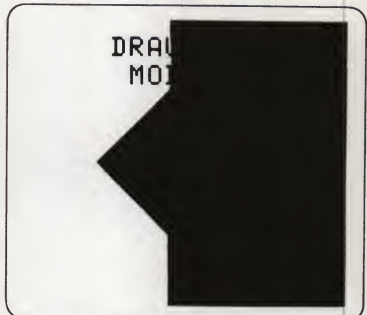
Mode 1 (Clear mode-erases the parts of the pattern in the original screen)



Mode 2 (Set mode-adds the pattern to the original screen)



Mode 3 (Complement mode-toggles the pattern over the original screen)



Mode 4 (Jam mode-the same as set mode on this terminal)

Figure D-2. Examples of Drawing Modes

CLEAR MODE. Clear mode causes selected display bits to be turned off. The "selected bits" are those that are "on" in the line pattern. If a solid line type (the default) has been selected, all of the bits in a vector will be selected. In clear mode this means that all of the dots making up a vector will be turned off. This allows you to draw dark vectors on a white background. Only those bits that are in the pattern are cleared. Bits that are off in the pattern do not affect the display.

SET MODE. Set mode is similar to clear mode except that the selected bits are turned on instead of off. Only the bits that are on in the line type are affected.

COMPLEMENT MODE. Complement mode causes the selected display bits to change state (on to off, off to on). Again only those bits that are on in the line type or pattern are affected.

JAM MODE. Jam mode functions identically to Set Mode on the HP150 system; it has been included for compatibility with other HP systems.

SELECTIVE ERASE. A vector drawn in set mode can be selectively erased by redrawing it in clear mode. This will cause gaps to occur if the erased line is intersected by other lines. This problem can be overcome by initially drawing the line in complement mode and then redrawing it in complement mode to erase the line. This technique will preserve the original display. Complement mode is useful for drawing and erasing temporary figures.

Example: Select complement mode, draw a vector, and then erase the vector by redrawing.

```
Ec * m 3A      (select complement mode)
Ec * p a 100,300 300,300Z    (draw vector)
Ec * p a 100,300 300,300Z    (erase vector)
```

Drawing Patterns

You can select the dot pattern used when drawing vectors or filling rectangular areas. Dotted and dashed lines can be drawn by selecting one of nine predefined line patterns or a user defined line or area pattern. This allows you to use different line patterns to distinguish between groups of plotted data or easily generate shading and cross hatching for use in engineering drawings, graphs or fabric patterns.

LINE TYPE. One of eleven line types can be selected. Once a line type has been selected, all drawing vectors are drawn using that line type. The patterns for the predefined line types are shown in figure D-3. Refer to the Define Line Pattern command for additional information.

Select Line Type: Ec * m <line type> b

where: <line type> is



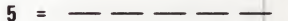

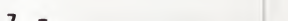

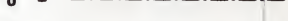


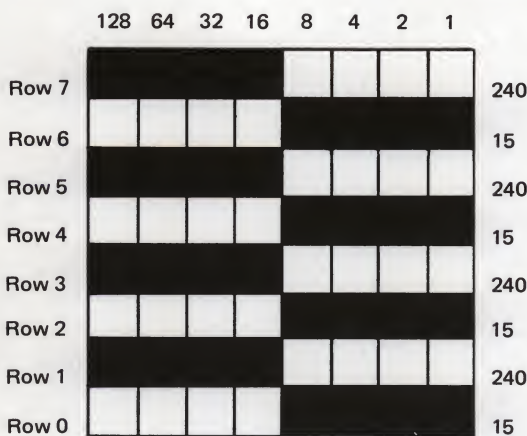
- | | |
|-----------------------------|----------------------------------------------------------------------------------------|
| 1 Solid line (default) | 1 =  |
| 2 User defined line pattern | |
| 3 Current area pattern | 4 =  |
| 4 Predefined pattern #1 | 5 =  |
| 5 Predefined pattern #2 | 6 =  |
| 6 Predefined pattern #3 | 7 =  |
| 7 Predefined pattern #4 | 8 =  |
| 8 Predefined pattern #5 | 9 =  |
| 9 Predefined pattern #6 | 10 =  |
| 10 Predefined pattern #7 | 11 =  |
| 11 Point plot | 11 = (POINT PLOT) |

Figure D-3. Line Drawing Patterns

Point plot causes a single point to be plotted at the coordinates specified by the data. This line type is useful for generating "scattergram" type graphs. If current area shading is selected (type = 3) the line patterns used are selected from the eight lines making up the area fill pattern (refer to Define Area Pattern). The display is divided into groups of eight rows and eight columns. Horizontal and vertical lines are drawn using the appropriate row or column from the area pattern. Diagonal lines are drawn using a solid vector. See the Illustration below.



(A) Area Fill Pattern



(B) Display

Figure D-4. Using Area Patterns as Line Types

Ec * m 240 15 240 15 240 15 240 15 D

Defines the user-definable fill pattern to that shown in Figure D-4A.

Ec * m 2 G

Sets the current area fill pattern to the user-defined area pattern

Ec * m 3 B

Sets the current line type to the current area fill pattern.

Ec * pa 200,150 300,150 300,250 200,250 200,150
300,250 Z

Draws the vectors shown in Figure D-4B.

NOTE

Only horizontal and vertical vectors can be defined with an area pattern. All diagonal vectors are drawn as a solid line.

Adjacent horizontal or vertical lines using the user defined line type (type = 2) can be used to create patterns more complicated than those available in an 8×8 area pattern. User defined line and area patterns are described in the following paragraphs.

USER-DEFINED LINE PATTERN. The dot pattern used to draw vectors can be defined programmatically. Once a pattern is defined, you must select the user defined line type (type = 2) using the Select Line Type command. Figure D-5 gives examples of line patterns.

A user-defined line pattern is composed of a dot pattern and a scale factor. The dot pattern is a sequence of eight 1's and 0's. Using the default drawing mode, points indicated as a "1" in the pattern are drawn using the primary pen, and the points indicated as a "0" are left unchanged.

The pattern is given as a decimal number between 0 and 255 that is the decimal equivalent of the 8-bit binary pattern. The default pattern is all "1"s (255). For example, ... = 10101010 (binary) = 170 (decimal). The actual number used for the pattern can be between -32768 and 32767. The least significant 8 bits of the number's 2's complement equivalent are used to determine the pattern.

The scale factor indicates how many times each bit in the pattern is repeated. The default scale factor is 1. For example, a scale factor of 3 applied to the pattern defined above results in a pattern of or 111000111000111000111000 (binary).

The command for creating a user-defined line type is:

```
Ec * m <pattern><scale>c
```

where <pattern> is an integer in the range (-32768 and 32767) and <scale> is an integer in the range (0 to 255 modulo 16).

Example: Define a pattern to generate the following vector:

```
11111111110011001111111111001100
pattern = 11111010 = 250
scale = 2
Ec * m 250 2 c
```

The result is displayed in the fourth user-defined line in Figure D-5 below.

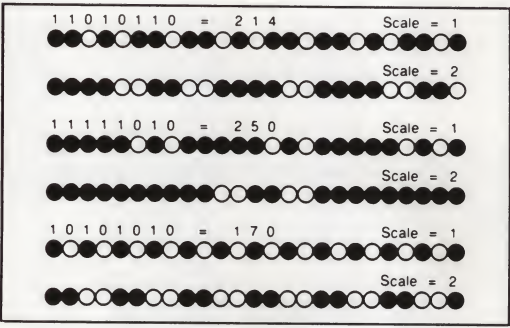


Figure D-5. Examples of User-Defined Line Types

Line patterns too complex to be obtained from an 8×8 area pattern can be generated by plotting a series of lines and varying the patterns used for successive lines. Complex patterns such as those used in weaving can be generated easily using this technique.

Area Fills

The terminal has two types of area fill specifications — rectangular and polygonal. An area can be filled with one of a variety of predefined patterns or with a user-defined pattern. The pattern can also be used to provide line patterns for horizontal or vertical lines when the area pattern is selected as the line type. (Refer to "Using Area Fills As Line Types.")

When an area fill pattern is selected, the entire screen is divided into 8×8 cells. Each location is mapped to the corresponding bit in the pattern. When an area fill operation is performed, the area fill pattern is duplicated to fill the area. The pattern starts at screen coordinates 0,0 (see Figure D-6 below).

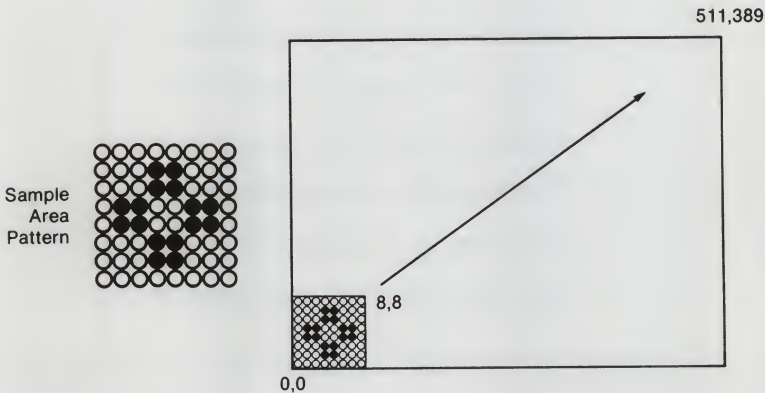


Figure D-6. How The Area Fill Pattern Is Mapped

Selecting An Area Fill Pattern

The default pattern for area fill is a user-defined pattern. To select the type of area fill pattern to be used, use the following escape sequence:

```
Ec * m <area pattern> g
```

where <area pattern> is selected from:

- 1 Solid area fill
- 2 User defined area fill pattern (default)
- 3 Pattern #0 (short dashed hatching)
- 4 Pattern #1 (long dashed hatching)
- 5 Pattern #2 (hatching)
- 6 Pattern #3 (cross hatching)
- 7 Pattern #4 (fine cross hatching)
- 8 Pattern #5 (medium checkerboard)
- 9 Pattern #6 (fine checkerboard, 1:1 blend)
- 10 Pattern #7 (3:1 blend)

User Defined Area Fill Patterns

The user area pattern is defined by 8 parameters, one for every row of dots in the pattern. Each parameter is an integer in the range -32768 to 32767. The number is interpreted as a 2's complement number, and the least significant 8-bits are used to obtain a value between 0 and 255. The 8-bit number (0 to 255) represents an 8-bit binary pattern. Bits set to 1 are drawn using the primary pen, and bits set to 0 (zero) are not drawn (depending on the current drawing mode).

The command for defining a user area fill pattern is:

```
Ec * m <row 0><row 1>. . .<row 7> d
```

where <row 0> is the 8-bit pattern for row 0

.

<row 7> is the 8-bit pattern for row 7

Example: Define the simple checkerboard pattern seen in Figure D-8 below.

```
Ec * m 170 85 170 85 170 85 170 85 D
      Row 0
                                     Row 7
```

NOTE

The scale factor of an area fill pattern is always 1. This resembles the HP 2627A but is unlike all other Hewlett Packard terminals.

Other examples of user defined area patterns are shown in Figure D-7 below.

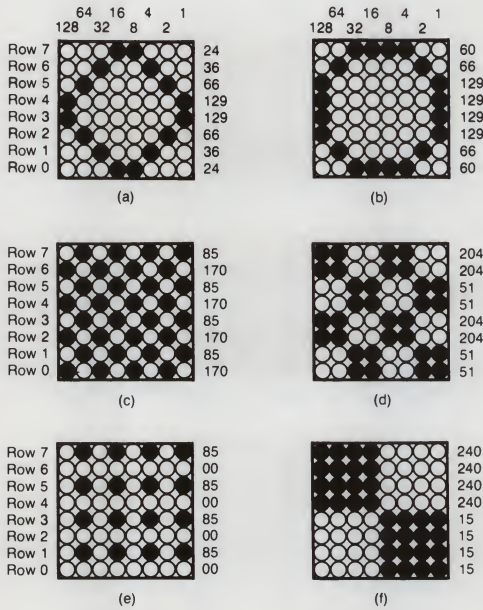


Figure D-7. Examples of User-Defined Area Fill Patterns

Using Area Fill Patterns as Line Types

If you select type 3 for the Line Type command, the line pattern is created from the current area fill pattern. Horizontal and vertical lines are drawn using the appropriate row or column from the area fill pattern. Diagonal lines are always drawn using a solid vector; they do not follow the area fill pattern. If a line is longer than 8 dots, the pattern is repeated to complete the vector.

Example: Plot three vectors (200,150), (300,150 . . .) using a user defined area fill pattern of 240,15,240,15 (Figure D-8 below).

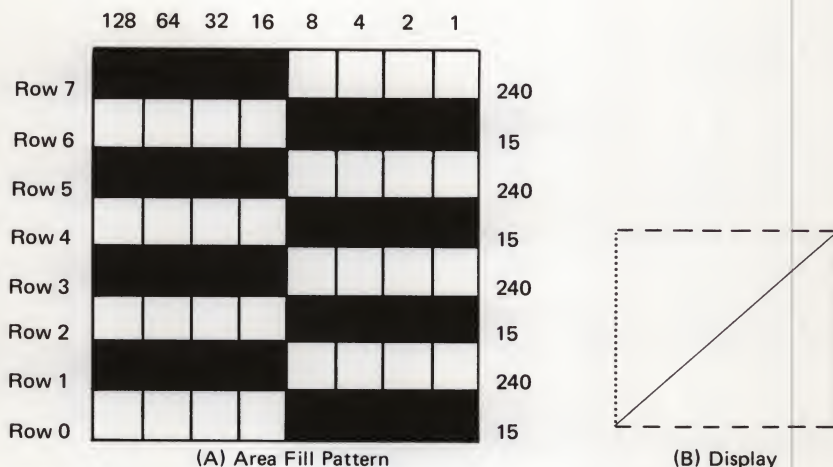


Figure D-8. Using Area Fill Patterns as Line Types

Ec * m 240 15 240 15 240 15 240 15 D

Defines the user-definable fill pattern to that shown in Figure D-8A.

Ec * m 2 G

Sets the current area fill pattern to the user-defined area pattern

Ec * m 3 B

Sets the current line type to the current area fill pattern.

Ec * pa 200,150, 300,150, 300,250 200,250 200,150
300,250 Z

Draws the vectors shown in Figure D-8B.

Rectangular Area Fills

A rectangular area can be filled in with a pattern simply by sending the lower left and upper right coordinates in an escape sequence. The coordinates can be either in absolute or relocatable ASCII format. The pattern used for area fill is determined by the Select Area Pattern command.

NOTE

The terminal can also fill irregular polygons. See "Polygonal Area Fills" in this section.

FILL RECTANGLE, ABSOLUTE:

```
Ec * m <x1><y1><x2><y2> e
```

where <x1><y1> are the absolute coordinates of the lower left corner of the rectangle area to be filled (-16384 to +16383), and <x2><y2> are the absolute coordinates of the upper right corner of the rectangular area to be filled (-16384 to +16383).

Example: Using area fill pattern 5, fill a rectangle defined by the diagonal 50,50 300,300.

```
Ec * m 5 g 50,50 300,300 E
```

FILL RECTANGLE, RELOCATABLE:

```
Ec * m <x1>,<y1><x2>,<y2> f
```

where <x1> and <y1> are the relocatable coordinates of the lower left corner of the rectangular area to be filled (-16383 to 16383), and <x2> and <y2> are the relocatable coordinates of the upper right corner of the rectangular area to be filled (-16383 to 16383).

This command is used in conjunction with the relocatable origin.

Example: Load a function key with the command:

```
Ec * m l 20,20 30,30 F
```

Use the cursor control keys to move the graphics cursor while periodically pressing this function key.

Polygonal Area Fills

Begin Polygon Area Fill: `Ec * p s`

Close Polygon/Begin New Polygon: `Ec * p a`

Close Polygon Area Fill: `Ec * p t`

You can define a polygon with as many as 105 sides and fill the shape with the current area fill pattern. The Begin Polygon Area Fill command causes subsequent coordinate pairs to be read as vertices of the polygon. When a lift pen command (`Ec * p a`) occurs in the middle of a polygon area fill sequence, a new polygon is started (see example). The Close Area Fill command (or any capital letter) causes the polygon to be filled using the current drawing mode and area pattern. Note that it is not necessary to specify a vector from the last point back to the first point; the polygon automatically closes itself at the end of the sequence.

If the polygon definition crosses over itself, the areas are defined in alternate order. See Figure D-9 and D-10 below.



Figure D-9. Overlapping Polygon Area Fills

Example: Move the pen to 33,0, and define and fill a pentagon 100 units on a side. Lift and move the pen 40 10, and define another pentagon inside the first pentagon.

```
Ec * p a s 33,0 133,0 166,95 83,150 0,95  
a 40,10 12,91 83,138 153,91 125,10 T
```



Figure D-10. Polygon Area Fill Example

NOTE

When using user-defined softkeys to define polygon area fills, the entire polygon specification must be contained within one softkey to avoid loss of data.

Area Boundary Pen

You can outline a filled area with a solid line by enabling the area boundary pen. When the area boundary pen is disabled, the edges of the filled area are the same pattern as the interior.

Set Area Boundary Pen: `Ec * m<boundary pen> h`

where `<boundary pen>` may be:

`0` = enable boundary pen

`No parameter` = disable boundary pen

Graphics Relocatable Origin

The relocatable origin is a means of adding an offset to the x and y coordinate values of data points. The value of the relocatable origin is added to the value of the x and y coordinates sent by the host computer.

The relocatable origin allows you to use one set of data and drawing commands to display a figure at several different positions. The value of the relocatable origin is added to the relocatable data to obtain the coordinates used to draw the figure.

The terminal provides three commands for setting the graphics relocatable origin:

- Set Relocatable Origin In Absolute Coordinates
- Set Relocatable Origin To Current Pen Position
- Set Relocatable Origin to Current Cursor Position

Set Relocatable Origin in Absolute Coordinates

`Ec * m<x>,<y>;`

where $-16383 < \langle x \rangle, \langle y \rangle < 16383$

This command sets the relocatable origin to the coordinates given in $\langle x \rangle$ and $\langle y \rangle$. The values of $\langle x \rangle$ and $\langle y \rangle$ are in the absolute data format. Once the relocatable offset has been added, the resultant data is stored in the terminal in absolute format.

`Ec * m-300 -100J`

`Ec * p ah300,100 500,100 500,200 300,200 300,100Z`

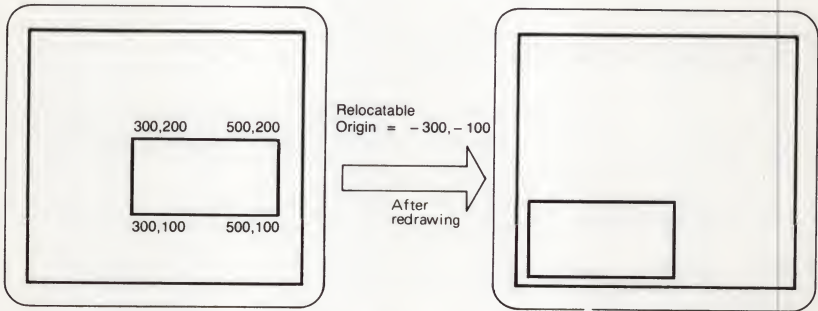


Figure D-11. Relocatable Origin.

Set Relocatable Origin To Current Pen Position

`Ec * mk` or `Ec * pe`

This command sets the relocatable origin to the current position of the graphics pen. This allows you to position the pen using absolute, incremental, or relocatable data and then plot a figure at that location using relocatable data.

Set Relocatable Origin To Current Graphics Cursor Position

`Ec * ml`

This command takes the absolute coordinates of the virtual graphics cursor position as the relocatable origin.

Selecting The Graphics Default Parameters

Graphics parameters can be set to their default (power-on or full reset) values (see Table D-5) by issuing the following sequence:

```
Ec * m <default flag> r
```

It may be necessary to reselect graphics settings before sending graphics data to the terminal. See the Status section for further information on graphics status requests.

Graphics Hard Reset

Graphics hard reset performs as if a hard reset were initiated for graphics only. It sets all graphics parameters to their default values as specified for `Ec * mr` (see Table D-5) plus the following:

1. Clears raster memory buffer
2. Drawing pen is positioned at location 0,0

A graphics hard reset can only be performed programmatically:

```
Ec * w r
```

Table D-5. Graphics Parameter Default Values.

PARAMETER	DEFAULT VALUE
**Pen Condition	down
**Line Type	1 (solid)
**Drawing Mode	2 (SET)
**User Line Pattern	255,1
**Area Fill Type	2 (user-defined pattern)
**User Area Fill Pattern	255,255,...,(solid)
**Boundary Pen	off
**Text Size	1
**Text Direction	1
**Text Origin	1 (left, bottom)
**Text Slant	0 (off)
**Graphics Text	off
Relocatable Origin	0,0
Graphics Video	on
Alphanumeric Video	on
Graphics Cursor	off
Graphics Cursor	
Address	0,0
Rubberband Line	off
Alphanumeric Cursor	on
Compatibility Mode	
Page Full Straps	0 (out)
GIN Strap	0 (CR only)

**If a "1" is used for the <default flag> (`Ec•mr`), only these parameters are defaulted. When no default value is used (`Ec•mr`), all parameters are defaulted.

Plotting Sequences

All vector plotting sequences are initiated by `Ec * p` Table D-6 lists the commands that can be used within a plotting sequence.

After `Ec * p` has been sent, the drawing format is normally specified before data is sent.

If no format is specified, ASCII absolute is assumed. There is no explicit draw vector command.

Plotting Commands

Graphic data is made up of vectors (line segments). There is no explicit "draw vector" command. Instead, the terminal uses the concept of a "pen" in drawing vector data.

The general format for a plotting sequence is:

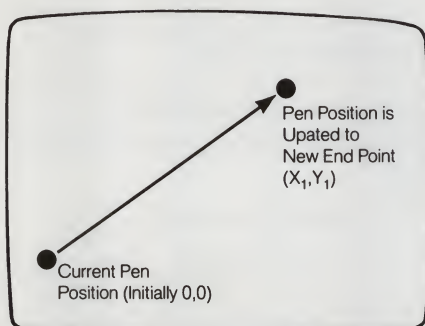
`Ec * p <pen state> <x1> , <y1> <x2> , <y2> . . . Z` (or any capital letter)

where `<pen state>` indicates whether the pen is up or down, and the values are delimited by spaces or commas. The capital "Z" (a non-operative) terminates the sequence.

Table D-6. Graphics Plotting Control Functions

Ec * p <parameters and data>	
PARAMETER	DESCRIPTION
a	lift the pen
b	lower the pen
c	use graphics cursor position as new point
d	draw a single dot at the current pen position
e	set relocatable origin = current pen position
f	use ASCII absolute format
g	use ASCII incremental format
h	use ASCII relocatable format
i	use binary absolute format
j	use binary short incremental format
k	use binary long incremental format
l	use binary relocatable format
z	NOP/synch

When enough parameter bytes have been received (the number depends on the parameter's format) to specify a data point, the pen is moved from its current position to the new end point. If the pen is down, a vector will be drawn. If the pen is up, the pen is moved to the new point (without drawing a vector) and lowered. In either case, the new point becomes the *current pen position*. The following illustration demonstrates the current pen position and the new end point.



Plotting sequences can extend indefinitely. In general, longer sequences are preferred as they minimize the overhead needed for a plot sequence. As the sequence length decreases, the percentage of prefix characters increases, and the drawing rate goes down. The worst possible case would be to send each vector in its own sequence; approximately 50% of the characters sent would be overhead, reducing vector speed by a factor of 2.

Note that if a parameter byte is lost or garbled in transmission, all following end points will be improperly read. To minimize data errors caused by the loss of a data byte, any command can be used to reset the parameter count and restore synchronization. Nops (z), redundant format, or pen down commands can also be inserted to insure synchronization if necessary.

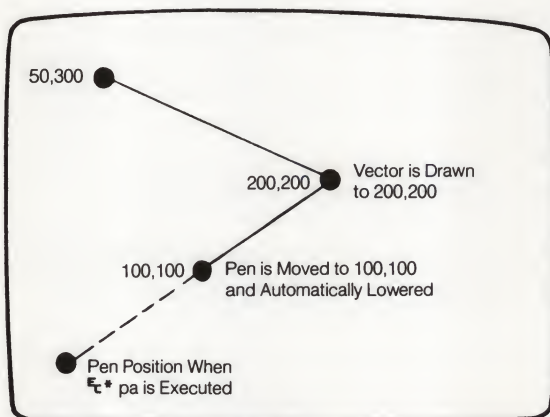
Lift Pen

`Ec * pa`

This command causes the imaginary plotting pen to be lifted from the drawing surface. Movement of the pen from the current position will not draw a line. The pen must be lowered (by supplying a coordinate or a lower pen command — `Ec * pb`) before a line can be drawn.

Example: Lift the pen, move it to 100,100, draw a vector to 200,200 and then to 50,300.

`Ec * pa 100,100 200,200 50,300Z`



NOTE

This command can also be used in conjunction with polygonal area fill commands. See "Polygonal Area Fills" in this chapter.

Lower Pen

`Ec * pb`

This command causes the imaginary plotting pen to be lowered to the drawing surface. Movement of the pen from the current position will draw a line.

Example: Draw a vector from the current pen position to 194,250.

`Ec * pb 194 250Z`

Use Cursor As Next Data Point

`Ec * pc`

This command causes the position of the graphics cursor to be used as the next data point.

Rubberband Line Mode

Turn Rubberband Line On: `Ec * dm`

Turn Rubberband Line Off: `Ec * dn`

"Rubberband Line" mode causes the terminal to display a temporary line connecting the graphics cursor to the current pen position. As the cursor is moved (using the cursor control keys or move cursor commands), the temporary line moves, stretches, or contracts as required to maintain the connection. The temporary line is "set" when the cursor position is entered as a new point by executing the `Ec * pc` command. The origin of the temporary rubberband line is then updated to the new point and the process can be repeated.

NOTE

If the graphics cursor is not already on, activating the rubberband line function turns on the graphics cursor.

Draw A Point At The Current Pen Position

Ec * pd

The command draws a point at the current pen position. The pen is set to up. This command is ignored if encountered during an area fill sequence.

Vectors

Graphics data is made up of vectors. Each vector is specified by the current graphic starting point and an end point. The current graphic starting point is one of the following:

0,0 Initial starting point

Last point defined by the graphics cursor (**Ec*pc**).

Last point defined by the data in a draw or move command

(**Ec * p f/g/h/i/j/k/l**).

Graphic points are specified in one of the following formats:

ASCII Absolute

ASCII Incremental

ASCII Relocatable

Binary Absolute

Binary Incremental

Binary Short Incremental

Binary Relocatable

If no format is specified in the graphic command, ASCII absolute format is assumed. More than one point can be given in a command. This minimizes communications overhead. Tables D-7, D-8, and D-9 provides a reference for computing data bytes used in the various vector formats.

ASCII Formats

In the ASCII formats, coordinates are specified with ASCII characters 0 through 9. This means that numeric characters generated by a simple print statement can be used to specify X,Y pairs. The first value is used as the X coordinate, and the second as the Y coordinate.

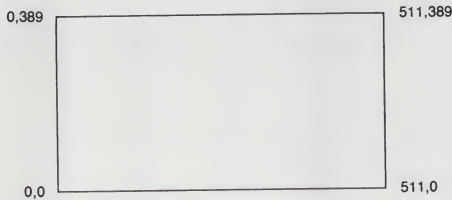
Spaces or commas must be used to delimit the X and Y values. Excess delimiters are ignored. Digits following a decimal point are ignored (i.e. 123.456 is read as 123).

NOTE

Exponential notation cannot be used. Consequently, the values must be in integer form. The number of bytes necessary to specify a single end point depends on the magnitude of the values.

ASCII ABSOLUTE FORMAT. The values used in the ASCII absolute format can range between -16384 and 16383. Note that only points where X is the range 0 to 511 and Y is the range 0 to 389 will be visible on the screen. The following example draws vectors around the perimeter of the screen:

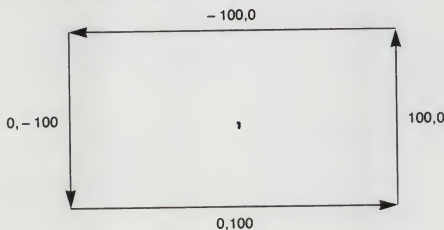
```
Ec * p a 0,0 511,0 511,389 0,389 0,0Z
```



Since no format is indicated, ASCII absolute is assumed. The "a" raises the pen, which is moved to (0,0) and lowered. Vectors are then drawn to (511,0), (511,389), (0,389), and back to (0,0). (Note that the values are delimited by spaces or commas. The upper case "Z" [a nop] terminates the sequence. Imbedded carriage return and line feed characters are ignored.)

ASCII INCREMENTAL FORMAT. In the ASCII incremental format you can specify a delta X and a delta Y. These values are added to the current pen position to obtain a new end point. The first value is read as delta X and the second as delta Y. For example to draw a square 100 units on a side, the following sequence could be used:

```
Ec * p g 100, 0 0, 100 -100, 0 0, -100 Z
```



Beginning at the current pen position, a series of vectors is drawn by moving the pen 100 units to the right, up 100 units, left 100 units, and finally down 100 units. The same figure could have been drawn at any screen location by first positioning the pen to the desired starting point before sending the drawing sequence.

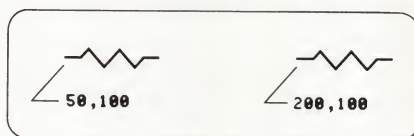
ASCII RELOCATABLE FORMAT. The ASCII relocatable format allows you to use a relocatable origin to be added to the incoming X and Y coordinate values. The resultant values are then treated as absolute coordinates by the terminal. The relocatable format allows you to use absolute data as if it were incremental by merely changing the relocatable origin. For example, symbol elements specified in absolute coordinates can be drawn in different locations as shown in the following example.

Example: Draw a resistor symbol stored in absolute coordinates at screen locations 50,100 and 200,100.

```
Resistor Data = 0,10
                10,10
                15,15
                25,5
                35,15
                45,5
                50,10
                60,10
                0,20    60,0
                0,0     60,0
```

```
⌘*m50,100J
⌘*pah0,10 10,10 15,15 25,5 35,15
                45,5 50,10 60,10Z

⌘*m200,100J
⌘*pah0,10 10,10 15,15 25,5 35,15
                45,5 50,10 60,10Z
```



Binary Format

In binary format all points are sent in a packed binary format. The coordinate values are sent using the bit patterns of the ASCII characters listed in Table D-7. The number of characters required to specify a coordinate depends on the format used. The Binary data can be long, short, or medium. The vectors are made up of 2, 4, or 6 bytes of coordinate information. The numbers are represented in a 5-bit per byte format. The values for X and Y coordinates can be from -16384 to 16383.

BINARY ABSOLUTE FORMAT. Binary absolute data is plotted with respect to an origin at 0,0. Four bytes are required to specify a single end point. A 10 bit coordinate in the range 0-1023, is sent for both x and y.

The bytes are ordered as follows:

BIT	7	6	5	4	3	2	1	
BYTE 1	0	1	X9	X8	X7	X6	X5	HI X
BYTE 2	0	1	X4	X3	X2	X1	X0	LOW X
BYTE 3	0	1	Y9	Y8	Y7	Y6	Y5	HI Y
BYTE 4	0	1	Y4	Y3	Y2	Y1	Y0	LOW Y

Although it is possible to send coordinates in the range 0 to 1023, only points in the range 0-511 for X, and 0-389 for Y are visible on the screen. Vectors going off the screen are clipped. If the data requires scaling, this must be done before the data is sent to the terminal.

The following example shows how the 4 data bytes are computed. The numbers are converted to the 10 bit binary equivalent. Bits 7 and 6 are set to 01 to indicate a parameter.

```

X = 0 = 00000 00000      Y = 0 00000 00000
      HI X  LOW X          HI Y  LOW Y

BYTE 1 = 01 00000 = SPACE HI X
BYTE 2 = 01 00000 = SPACE LOW X
BYTE 3 = 01 00000 = SPACE HI Y
BYTE 4 = 01 00000 = SPACE LOW Y

X = 360 = 01011 01000     Y = 180 = 00101 10100
      HI X  LOW X          HI Y  LOW Y

BYTE 1 = 01 01011 = + HI X
BYTE 2 = 01 01000 = 0 LOW X
BYTE 3 = 01 00101 = 0 HI Y
BYTE 4 = 01 10100 = 4 LOW Y

```

An escape sequence to draw a vector from 0,0 to 360,180 is as follows:

```

Ec * p i a SP SP SP SP + 0/0 4 Z
      X=0/      Y=0/      X=360/      Y=180/

```

Ec * p selects a plotting sequence. The "i" specifies absolute format. The "a" raises the pen up. The first 4 bytes (all spaces) move the raised pen to 0,0 where it is lowered. The next 4 bytes specify the point 360,180. After the 4th byte is received, the pen is moved to that point, drawing a vector. The upper case "Z" terminates the escape sequence. Note that if spaces are used in the data sequence they are interpreted as data and could result in an improper plot.

BINARY SHORT INCREMENTAL FORMAT. The short incremental format uses two bytes to specify a delta X and a delta Y in the range -16 to +15. The five least significant bits are interpreted as a signed, two's complement number. This number is added to the current pen position to obtain the new end point. The data bytes are ordered as follows:

BIT	7	6	5	4	3	2	1
BYTE 1	0	1	< DELTA X				>
BYTE 2	0	1	< DELTA Y				>

The following example illustrates the computation and use of the short incremental format:

```

DELTA X = -12 = 10100   DELTA Y = 6 = 00110
BYTE1 = 01 10100 = 4   DELTA X
BYTE2 = 01 00110 = 6   DELTA Y

```

The following sequence moves the pen to 360,180 in absolute format, then draws a vector to $X = 360 - 12 = 348, y = 180 + 6 = 186$.

Ec * p i a + (0/0 4 j 4 & <byte1><byte2>...Z

BINARY INCREMENTAL FORMAT. Incremental is similar to short incremental, but with a larger range. Using six bytes, delta X and Y can range from -16384 to +16383.

BIT	7	6	5	4	3	2	1	
BYTE1	0	1	DX14	DX13	DX12	DX11	DX10	HI DELTA X
BYTE2	0	1	DX9	DX8	DX7	DX6	DX5	MID DELTA X
BYTE3	0	1	DX4	DX3	DX2	DX1	DX0	LOW DELTA X
BYTE4	0	1	DY14	DY13	DY12	DY11	DY10	HI DELTA Y
BYTE5	0	1	DY9	DY8	DY7	DY6	DY5	MID DELTA Y
BYTE6	0	1	DY4	DY3	DY2	DY1	DY0	LOW DELTA Y

The following sequence moves the pen to 360,180 in absolute format, then draws a vector to $X = 360 - 12 = 348, y = 180 + 6 = 186$.

```
Ec * p i a + ( 14 j 4 & <byte1><byte2>...Z
```

The following example shows how incremental data bytes are generated.

```

DELTA X = -400 = 11111 10011 10000
                  HI DX  MID DX  LO DX

DELTA Y = 100 = 00000 00011 00100
                  HI DY  MID DY  LO DY

BYTE 1 = 01 11111 = ?      HI DELTA X
BYTE 2 = 01 10011 = 3      MID DELTA X
BYTE 3 = 01 10000 = 0      LO DELTA X
BYTE 4 = 01 00000 = space  HI DELTA Y
BYTE 5 = 01 00011 = #      MID DELTA Y
BYTE 6 = 01 00100 = $      LO DELTA Y

```

Table D-7. Characters Used in Packed Data Formats

ASCII Character	Bit Pattern	ASCII Character	Bit Pattern
SP	01 0 0000	0	01 1 0000
!	01 0 0001	1	01 1 0001
"	01 0 0010	2	01 1 0010
#	01 0 0011	3	01 1 0011
\$	01 0 0100	4	01 1 0100
%	01 0 0101	5	01 1 0101
&	01 0 0110	6	01 1 0110
'	01 0 0111	7	01 1 0111
(01 0 1000	8	01 1 1000
)	01 0 1001	9	01 1 1001
*	01 0 1010	:	01 1 1010
+	01 0 1011	;	01 1 1011
,	01 0 1100	<	01 1 1100
-	01 0 1101	=	01 1 1101
.	01 0 1110	>	01 1 1110
/	01 0 1111	?	01 1 1111

BINARY RELOCATABLE FORMAT. Binary relocatable format specifies absolute X and Y coordinates in the range -16384 to +16383 using 6 bytes. The value specified in the relocatable origin command is taken to be the 0,0 point. The actual screen address is computed by the terminal by adding the relocatable origin to the X,Y pair.

```

      BIT   7   6   5   4   3   2   1
BYTE 1  0   1  X14 X13 X12 X11 X10  HI   X
BYTE 2  0   1  X9  X8  X7  X6  X5   MID  X
BYTE 3  0   1  X4  X3  X2  X1  X0   LOW  X

      BIT   7   6   5   4   3   2   1
BYTE 4  0   1  Y14 Y13 Y12 Y11 Y10  HI   Y
BYTE 5  0   1  Y9  Y8  Y7  Y6  Y5   MID  Y
BYTE 6  0   1  Y4  Y3  Y2  Y1  Y0   LOW  Y

```

The following example shows how relocatable data bytes are computed.

```
RELOC X = -600 = 11111 01101 01000
                    HI X  MID X LOW X
```

```
RELOC Y = 200 = 00000 00110 01000
                    HI Y  MID Y LOW Y
```

```
BYTE 1 = 01 11111 = ?      HI X
BYTE 2 = 01 01101 = -      MID X
BYTE 3 = 01 01000 = (      LOW X
```

```
BYTE 4 = 01 00000 = space  HI Y
BYTE 5 = 01 00110 = &      MID Y
BYTE 6 = 01 01000 = (      LOW Y
```

Mixing Data Formats

There are no restrictions on mixing data formats. Simply specify the new format to be used and follow it with data in the new format. Note that by restricting data values to binary values between 32 and 63, the printing graphics characters and numbers, the plotting commands "a-z" can be intermixed in the binary data.

Example: Move the pen to 360,180 in ASCII absolute format, then draw a box 10 units wide by 5 units high using binary short incremental.

```
Ec * p a f 360,180 j * SP SP 0/0 6 SP SP ; Z
          1      2      3      4 5
```

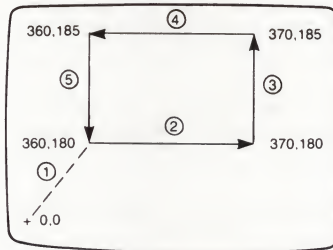


Figure D-12. Example of Mixed Data Formats

Table D-8. Absolute Format Addressing Bytes

	0	1	2	3	4	5	6	7	8	9
350	<+>	+?	+■	+!	+*	+●	+§	+%	+&	+'
360	+(>	+<	+■	+!	+*	+●	+§	+%	+&	+'
370	+2	+3	+4	+5	+6	+7	+8	+9	+	:
380	+<	+>	+>	+>	+■	+!	+*	+●	+§	+%
390	+&	+	+	+	+	+	+	+	+	+
400	0	1	2	3	4	5	6	7	8	9
410	:	:	:	:	:	:	:	:	:	:
420	-&	-%	-&	-!	-*	-●	-§	-%	-&	-'
430	-<	->	-0	-1	-2	-3	-4	-5	-6	-7
440	-8	-9	-:	-:	-<	->	->	->	-■	-!
450	."	."	."	."	."	."	."	."	."	."
460	."	."	."	."	."	."	."	."	."	."
470	."	."	."	."	."	."	."	."	."	."
480	/"	/"	/"	/"	/"	/"	/"	/"	/"	/"
490	/>	/>	/>	/>	/>	/>	/>	/>	/>	/>
500	/4	/5	/6	/7	/8	/9	/:	/:	/<	/>
510	/>	/7	0■	0!	0*	0●	0§	0%	0&	0'
520	0(>	0(>	0* >	0* >	0* >	0* >	0* >	0* >	0* >	0* >
530	02	03	04	05	06	07	08	09	0:	0:
540	<■	0*	0*	0*	0*	1■	1!	1*	1*	1*
550	1&	1'	1(>	1(>	1*	1*	1*	1*	1*	1*
560	10	11	12	13	14	15	16	17	18	19
570	1:	1:	1<	1*	1*	1*	1*	1*	1*	1*
580	2&	2*	2*	2*	2(>	2*	2*	2*	2*	2*
590	2,	2,	20	21	22	23	24	25	26	27
600	28	29	2:	2:	2<	2*	2>	2?	3■	3!
610	3*	3*	3*	3*	3*	3*	3*	3*	3*	3*
620	3,	3,	3,	3,	30	31	32	33	34	35
630	36	37	38	39	3:	3:	3<	3*	3*	3*
640	4■	4!	4*	4*	4*	4*	4*	4*	4(>	4(>
650	4*	4*	4,	4,	4,	4/	40	41	42	43
660	44	45	46	47	48	49	4:	4:	4*	4*
670	4>	4?	5■	5!	5*	5*	5*	5*	5*	5*
680	5(>	5(>	5*	5*	5,	5,	5/	50	51	51
690	52	53	54	55	56	57	58	59	5:	5:
700	5<	5*	5>	5?	6■	6!	6*	6*	6*	6*
710	6&	6'	6(>	6(>	6*	6*	6,	6,	6,	6/
0	■	!	■	■	■	■	■	■	■	■
10	■	■	■	■	■	■	■	■	■	■
20	■	■	■	■	■	■	■	■	■	■
30	■	■	■	■	■	■	■	■	■	■
40	!	!	!	!	!	!	!	!	!	!
50	12	13	14	15	16	17	18	19	!	!
60	1<	1*	1*	1*	1*	1*	1*	1*	1*	1*
70	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
80	10	11	12	13	14	15	16	17	18	19
90	1:	1:	1<	1*	1*	1*	1*	1*	1*	1*
100	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
110	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
120	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
130	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
140	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
150	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
160	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
170	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
180	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
190	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
200	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
210	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
220	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
230	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
240	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
250	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
260	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
270	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
280	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
290	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
300	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
310	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
320	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
330	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*
340	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

Note: ■ indicates a "space" character; every coordinate address must consist of the two characters shown in the table.

Table D-9. Incremental (short) Vector Bytes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
!	"	●	§	%	&	'	()	*	+	,	-	.	/	
-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
0	1	2	3	4	5	6	7	8	9	:	;	<	*	>	?

Graphics Functions In Display Functions Mode

The DISPLAY FUNCTNS key (at the MODES level) can be used to display the graphics escape sequences or the action of graphics control keys. The control sequences are entered into the alphanumeric display each time a command is executed. Table D-10 lists the graphics control sequences that are generated when DISPLAY FUNCTIONS is on.

Table D-10. Graphics Control Sequences Used in Record Operations

Key	Sequence	Description
↑ → ← ↓	none	Graphics cursor controls
Cursor fast	none	Graphics cursor fast
Graph cursor	Ec * dK Ec * dL	on off
Graph display	Ec * dC Ec * dD	on off
Graph clear	Ec * dA	
Alpha display	Ec * dE Ec * dF	on off

Figure D-13 shows the sequences generated when drawing a simple box. The graphics cursor is initially on and positioned at 0,0.

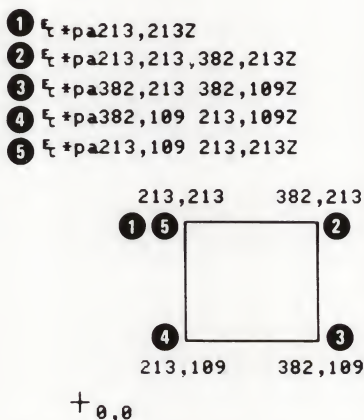


Figure D-13. Displaying Graphics Sequences

Graphics Hardcopy Operations

There are two methods of obtaining a hardcopy of the contents of graphics memory. One method uses the function keys and graphics keys on the keyboard. The other method uses escape sequences which may be coded into a program running on a host computer. The hardcopy may be output from the original internal printer, or from an external printer capable of interpreting raster escape sequences connected to the external printer port at the rear of the terminal.

When the optional internal printer is selected as the output device, a dot-by-dot transfer of the graphics memory is made on thermal paper. The image is 14.5 cm by 11 cm (5.75 by 4.25 inches) centered on the paper. The transfer takes approximately 30 seconds.

The external printers that can be used for graphics hardcopy are the HP 2671G, HP 2673A, HP 2932A, HP 82906A, and HP 2934A.

Initiating A Transfer From The Keyboard

The **Graph copy** key on the graphics/numeric pad initiates the graphics transfer. The keyboard must be in graphics mode for the **Graph copy** function to be performed. Pressing **CONTROL**, and the **⇧** key on your graphics keypad toggles the function of the keypad between graphics mode and numeric mode. Pressing the **Graph copy** key generates an error message if no valid destination device has been specified.

The destination(s) may be selected by pressing **User System**, **device control**, and to **devices**. You may then select **SERIAL DEVICE**, **INTERNAL PRINTER** and/or **HP-IB DEVICE**. **TO SERIAL DEVICE** selects the external printer port, and **TO INTERNAL PRINTER** selects the optional internal printer. **HP-IB DEVICE** selects an external printer on the HP-IB interface.

Using The Ec & p Escape Sequences

Coding a program to transfer graphics data either to the internal printer or to the external printer, or both, requires selecting the graphics memory as the source of either or both printers as the destination.

```
graphics memory as the source:      7s
internal printer as the destination: 6d
external printer as the destination: 4d
alternate printer as the destination: 5d
```

Example: Define graphics memory as source and internal printer as destination.

Ec & p 7s 6D

Example: Define graphics memory as source and both printers as destination.

Ec & p 7s 4d 6D

After the source and destination are defined, the transfer is initiated by either:

Ec & p F (copy file from source to destination)
or
Ec & p M (copy all from source to destination)

Note that an escape sequence is terminated with an uppercase character. Also, you may combine the source and destination assignments and the transfer initiation in one escape sequence:

Ec & p 7s 4d F

NOTE

The escape sequence (Ec & p 5 d) assigns the alternate printer as the destination for graphics data transfer. It also gives the host system control of the setting until the terminal receives the sequence Ec & p 3 D from the host computer or a HARD RESET from the keyboard or host. You should code the sequence Ec & p 3 D after graphics transfer is complete. This prevents the display of an I/O error message on the terminal screen when the keyboard operator tries to select \TO SERIAL DEVICE\ on the terminal configuration menu.

Graphics Text

Text strings can be written directly into the graphics image memory. An internal character generator converts the ASCII codes into a dot matrix representation which is drawn as vectors. The character set includes upper and lower case (95 characters) and the national characters. The characters will be drawn as a 5 by 7 matrix in a 7 by 10 cell, with descenders for lower case. This character set is in addition to the normal alphanumeric character set. While this character set may seem redundant, it offers the following advantages:

- Characters can be drawn at any dot position, rather than 24 by 80 alphanumeric character positions.
- Characters can be rotated in multiples of 90 degrees.
- Characters can be scaled in size, from 1 to 8 times.
- Characters can be slanted 45 degrees for an italics-like effect.
- Lines of characters can be right, left, or center justified.

Figure D-14 shows the graphics character set.

abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUVWXYZ

0000000000000000000000000000000000000000000000000000000

! " # \$ % & ' () * + , - . / 0 1 2 3 4 5 6 7 8 9 :

Journal of Interpersonal Violence 27(10)

• $\chi^2 = 2.0$ [1° 13']

£ 8 30030 ←

Foreign Characters

NOTE: All of the U.S. ASCII and foreign characters are accessible by entering "YES" in the ASCII 8 Bits config field of the Terminal Configuration menu; then entering foreign character mode set by typing **CTRL** , **Q** . You may shift back to the Roman base set by typing **CTRL** , **P** .

Figure D-14. Graphics Text Characters





Keyboard Control Of Graphics Text

Graphics text can be entered directly from the keyboard. The backspace, carriage return, and line feed functions work as expected (even on inverted text), making it easy to add or edit titles and labels. A summary of escape sequences and keyboard operations affecting Graphics Text Mode is given Table D-11.








NOTE

Use of keyboard functions may result in graphics text being overlaid.

Table D-11. Graphics Text Functions

Key	Description
Ec * d S	Selects the graphics image memory as the destination for all text. Characters entering from the keyboard or datacomm, are drawn as vectors in the graphics memory using the current text size and angle (see below). The graphics cursor indicates the position of the next character. Moving the graphics cursor will cause the next text line to begin at the new cursor position. The carriage return, line feed, and backspace functions work normally.
Ec * d T	Terminates Text Mode.
Ec * m <size> M	Increases the character size from 1 to 8X. The smallest character is a 5 by 7 matrix in a 7 by 10 cell. Increasing the size makes the dots bigger while the character is still drawn as a 5 by 7 matrix.
Ec * m <orienta- tion> N	Sets the character orientation (multiples of 90 degrees).
Ec * O Ec * P    	Turns slant on or off. Spaces one graphics text character to left. Spaces one graphics text character to right. (Vertical Tab). Spaces one graphics text line up. (The actual direction of movement will depend on the text orientation.)

The following keys function in the same manner as for alphanumeric text:

, , , , , , 

Program Control Of Graphics Text

All of the parameters for graphics text can be set programmatically. Commands are of the form: `Ec * m <parameter> <command>`. The command can be alone or part of another `Ec * m` sequence.

SIZE. The ASCII characters 1 through 8 specify the character size for graphics text (see Figure D-15). A "1" indicates the smallest character, a 5 by 7 dot matrix in a 7 by 10 cell. Increasing the size increases the size of the characters.

Set Graphics

Text Size: `Ec * m <size> m`

TEXT DIRECTION. This command uses the ASCII characters 1 through 4 to specify the text orientation (see Figure D-16). This also changes the direction of line feed, carriage return, and backspace.

- 1 — Normal (upright, the default)
- 2 — Rotated 90 degrees counter clockwise
- 3 — Rotated 180 degrees counter clockwise (inverted)
- 4 — Rotated 270 degrees counter clockwise

Set Graphics

Text Orientation: `Ec * m <orientation> n`

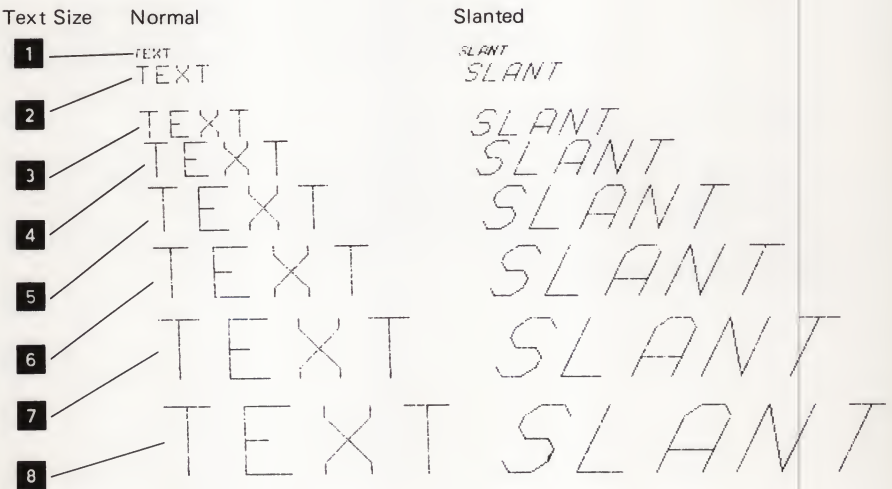


Figure D-15. Graphics Text Sizes

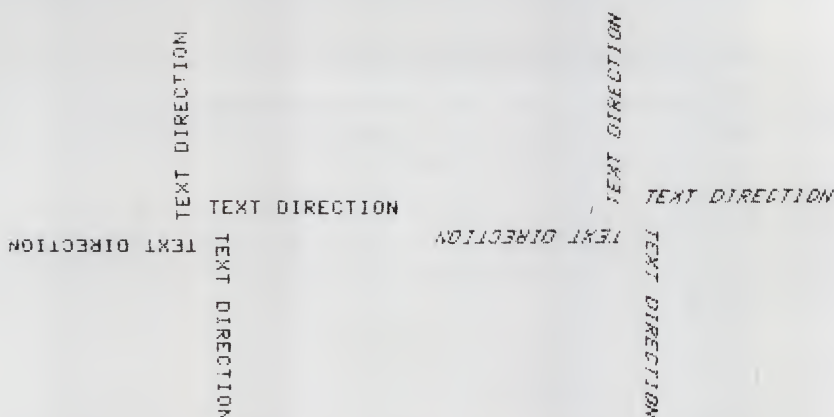


Figure D-16. Graphics Text Directions

SLANT. The graphics text characters can be slanted 45 degrees for an italics effect.

Turn On Graphics

Text Slant: `Ec * m o`

Turn Off Graphics

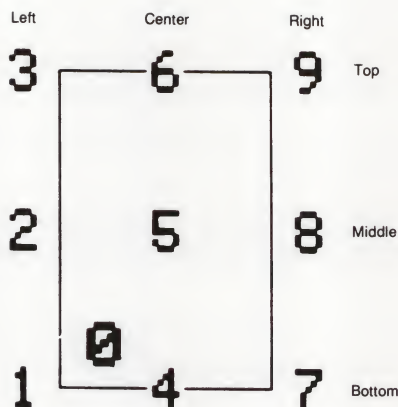
Text Slant: `Ec * m p`

JUSTIFICATION/ORIGIN. Text strings can be automatically right or left justified, or centered about a specified point. An ASCII character 0 through 9 indicates the origin (justification and base line) for characters with respect to the current pen position. This function is useful when drawing labels. (Refer to the Label command.)

Set Graphics

Text Justification: `Ec * m <origin> q`

If text is left justified, the current pen position is the left margin. Center causes the label to be centered on the pen position. Right justify selects the pen position as the right margin. Bottom, middle, and top select the base line for the line of text.



The numbers 1-9 represent the cursor position with respect to the character cell used for graphics text characters.

Figure D-17. Graphics Text Cursor Position

For example, if text was to be right justified and set with a base line on top of the normal character position, the number "9" would be used. Figure D-16 illustrates the various text positions.

When centering or right justification is used, the text strings are buffered (stored) until all of the characters in the string have been received. The string end is detected by a CR or LF. The string is not displayed until the CR or LF is received. This may be confusing when entering text from the keyboard. The maximum length of a string when center or right justifying is 73 characters (not including the CR(LF)). In all cases, data written beyond the edge of the screen is lost. There is no automatic RETURN when the screen boundary is reached.

TURNING GRAPHICS TEXT ON AND OFF. Graphics text mode can be turned on or off from a program. These two commands use the `E c * d` sequence but are discussed here under graphics text for completeness.

On. This command will cause Graphics Text Mode to be turned on. All displayable characters will be stored in the graphics memory. The current drawing mode remains in effect until a command is issued to change modes.

Text is drawn using the current text assignments for size and orientation. Graphics text mode accepts CR, LF, BS, HT, and VT as control characters. The `→`, `←`, `↑`, `↓` keys can be used to position the graphics cursor in character increments.

Turn On Graphics

Text Mode: `Ec * ds`

Turn Off Graphics

Text Mode: `Ec * dt`

If the graphics cursor is moved, the graphics text margin is moved to the new cursor or pen position.

Characters are drawn using the current drawing mode (set, clear, or jam). Entering a character, backspacing, and entering a second character causes an overstrike.

If a lower case "s" is used, additional escape parameters can be appended to the sequence. Otherwise the next characters will be routed to the graphics memory.

Examples:

`Ec * ds k 100,100 o B`

`Ec * d S This is a text string`

Off. This sequence turns off graphics text mode and restores normal alphanumeric operation.

Turn Off Graphics

Text Mode: `Ec * dt`

Note that the **ENTER** key and modify mode do not work on text in graphics mode.

GRAPHICS TEXT STATUS. You can check the current text settings with a graphics text status request. Refer to the Status section in this chapter for additional information.

LABEL. This sequence is used to send a single record of graphics text to the terminal. The characters are stored in the graphics memory using the current text size, angle, slant, and justification. The label is drawn beginning at the current pen position.

Graphics

Text Label: `Ec * l <text string> <CR> <LF>`

The record must end with a CR, LF, or both. A CR moves the pen to its original position when the label command was the first received. An LF moves the pen down one line (character spacing). Note that the actual directions moved following a CR or LF depend on the text orientation selected.

The maximum record length is 73 characters, not including the `Ec * l` preamble or the `<CR> <LF>`

Example: `Ec * l This is a sample label <CR> <LF>`

Compatibility Mode

Compatibility Mode allows the terminal to plot data intended for a terminal using a display with 1024 by 1024 addressable points or 4096 by 4096 (4014 emulation). This mode makes it possible to use graphics programs developed for use with other graphics terminals with a minimum of reprogramming.

The terminal operates in two submodes while in Compatibility Mode. In Alphanumeric mode the terminal simply displays alphanumeric data on the screen as in normal operation. In Graphics mode the terminal responds to alphanumeric data as vector coordinates. Normally the terminal will be switched between these modes to display messages, plot graphics figures, and then display additional messages. These modes are controlled with several control sequences. (These sequences are ignored or acted on differently if the terminal is not set for Compatibility Mode.)

Table D-12 lists the terminal's responses to Compatibility Mode control sequences.

If delays are required, the baud rate can be lowered or fill characters added to prevent data loss when operating the terminal at high speeds.

Vectors are drawn using the current line type, color pen, and line drawing mode. This gives you the capability of drawing dotted and dashed lines, etc. by changing the program to send the additional escape sequences.

Table D-12. Compatibility Mode Control Sequences

CONTROL SEQUENCE	DESCRIPTION	RESPONSE
ESC ESC	Read status and alpha cursor position	<status byte> <HI X> <LO X> <HI Y> <LO Y> <terminator>
<div> <div> <div>1 0 1 1 0/0 0/1 0/1 1</div> <div> <div>Depends on parity</div> <div>Hard Copy Unit 1 = not ready 0 = ready</div> </div> <div> <div>Auxiliary Device (inactive)</div> <div>Margin 0 = margin 1 1 = margin 2</div> <div>Mode 1 0 = Graphics Mode 0 1 = Alpha Mode</div> </div> </div> <p>The terminal will return one of the following characters as the status byte:</p> <div> <div>Printer ready</div> <div> + > % ' </div> </div> <div> <div>No printer or printer not ready</div> <div> : — Margin 2, Graphics Mode 9 — Margin 1, Graphics Mode 7 — Margin 2, Alpha Mode 5 — Margin 1, Alpha Mode </div> </div> </div>		
ESC H (20 ms delay) ESC ESC ESC H ESC ESC ESC F ESC ESC ESC ESC ESC ESC	Read graphics cursor position Read graphics cursor position when key struck Make hardcopy End graphics mode, clear screen, and home cursor Go into graphics mode (draw vectors) Go into alpha mode Backspace (H) Moves 1 space left (14 units) Horizontal Tab (I) Moves 1 space right (14 units) End graphics mode Line Feed (J) Moves 1 line down (22 units) Vertical Tab (K) Moves 1 line up (22 units)	<HI X> <LO X> <HI Y> <LO Y> <terminator> <KEY> <HI X> <LO X> <HI Y> <LO Y> <terminator>
<p>NOTES</p> <p>The terminal will normally respond with an ESC character when an ESC character is received. Compatibility Mode disables the terminal's ESC/ESC handshake. Compatibility Mode causes most control codes to be ignored.</p> <p>The Read Status, alpha cursor position, and graphic cursor position cause block transfers to the computer system. If the computer system does not use the DC1/DC2/DC1 handshake, I nhHndShk (G) and I nhDC2 (H) in the Terminal Configuration menu must be "YES" for these transfers to occur. (Refer to "Terminal Configuration Menu" in Section II)</p>		

Compatibility Mode Configuration

The five operating states for Compatibility Mode — OFF, UNSCALED, SCALED, SCL 4014, and UNSC 4014 — appear in the **GraphCompat** field in the Terminal Configuration menu. The default state is OFF. This field can be set programmatically using the “Ec & s” and “Ec * t” sequences shown in Table D-13. The P and Q straps and the Graph Compat parameter in the terminal configuration menu determine the terminal’s mode of operation after being initialized (power up or hard reset). The straps are interpreted as follows:

Table D-13. Commands for Selecting Compatibility Mode

Ec & s<x>p	Ec & s<y>q	Ec * t<z>d	Graph Compat
0	0	-	OFF
0	1	0	UNSCALED
0	1	1	UNS 4014
1	0	0	SCALED
1	0	1	SCL 4014
1	1	-	OFF

Table D-13 lists the escape sequences that set and clear the P and Q straps, and the graphics compatibility parameters in the terminal configuration menu. In addition, when in Compatibility Mode, you can select the following optional capabilities:

GRAPHIC INPUT TERMINATOR. You can select the terminator sent by the terminal following the input of cursor address information. The terminator can be CR, CR and EOT, or no terminator.

PAGE FULL BUSY. When this strap is in, the keyboard will be locked after the 35th line of text is received from the computer. The terminal can be cleared by pressing the **Graph clear** key. This strap is ignored in Unscaled Mode.

PAGE FULL BREAK. When this strap is in, the terminal will send a 200ms break signal to the computer after the 35th line of text is displayed. The terminal may also be set to BUSY (see Page Full Busy). When out, the strap will cause the cursor to home and the next 35 lines of text to be set with a left margin at x=259. This strap is ignored in Unscaled Mode. The commands to control these strap options are listed in Table D-13, above.

Graphics Data

4010 EMULATION. The terminal normally allows you to address 512×390 points; in Compatibility Mode, the number of addressable points extends to 1024×780 , emulating the 4010. Line length in normal operations is 24 lines by 80 characters; while in Compatibility Mode, line length is 35 lines by 74 characters (see Figure D-18). 4010-style graphics can be drawn either scaled or unscaled. Scaling divides X and Y coordinates by 2, mapping the 1024×780 display into 512 by 390. This allows a program written for the 1024×780 terminal to run unchanged, and still display the entire picture, with some loss in resolution (see Figure D-19).

Unscaled mode displays a 512 by 390 subset of the 1024×780 picture. The area this covers can be changed by modifying the value of the relocatable origin (and redrawing the picture). The relocatable origin is subtracted from all incoming coordinates in unscaled mode. If this is set to 0,0 (the default) the range $X = 0$ to 511, $Y = 0$ to 389 is displayed (see Figure D-18a).

Setting the origin to 0,360 would cover the $X = 0$ to 511, $Y = 360$ to 749. To display an area larger than 512×390 , you must change the scaling statements in the program.

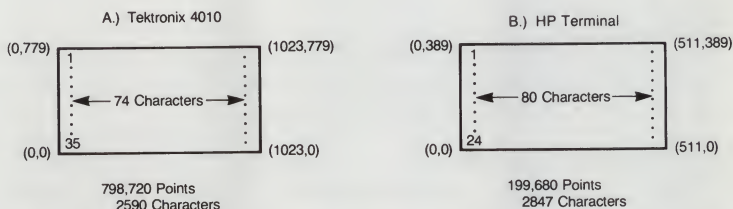


Figure D-18. Comparison of the 4010 and the HP Terminal

Graphics Data Format

In Compatibility mode the graphics data is formatted as two-byte coordinate values. The lower five bits of each byte are used to make a 10 bit (0-1023) coordinate. Data sent to the terminal must have the "Y" coordinate sent first; <Upper Y> <Lower Y> <Upper X> <Lower X>.

When data is returned to the computer (cursor position, etc.), the X coordinate is returned first; <Upper X> <Lower X> <Upper Y> <Lower Y>.

Data bytes sent to the terminal use bits 6 and 7 to indicate the byte is an Upper byte, a lower Y, or a lower X. Bit 8 (parity) is not used.

Bits		
7	6	
0	1	Upper X or Y byte
1	0	Lower X byte
1	1	Lower Y byte

These identifying bits allow you to send only the changed portion of a four byte address. The following data bytes must always be sent:

- Lower X byte
- Any changed byte
- Lower Y byte if the Upper X byte has changed

Table D-14 at the end of this section can be used to determine address bytes. For example, to plot the points A (0,0), B (0,31), C (256,31), D (256,0) the sequence shown in figure D-21 is used:

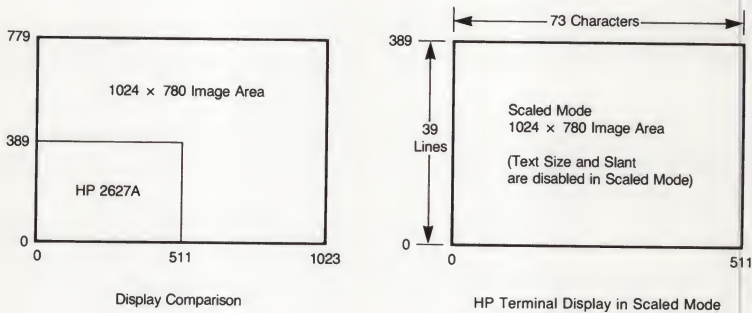
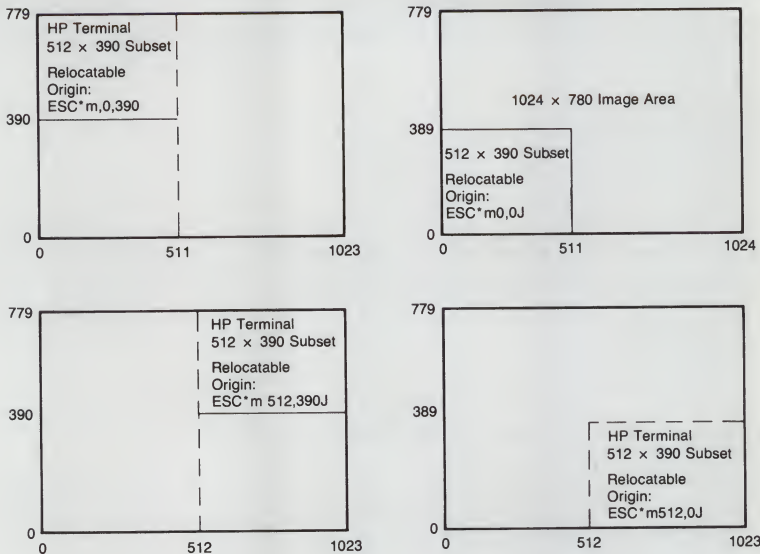


Figure D-19. Scaled Data



Changing the Relocatable Origin (ESC *m x,y J) on HP Terminal's Display
to Cover the Entire 1024 x 780 Display in Unscaled

Figure D-20. Unscaled Data

Text

When text is written to the graphics memory, the graphics cursor is moved to indicate where the next character will be stored. This differs from terminals that have only one mode for text and display the graphics cursor only when waiting for graphic input from the user.

SCALED MODE GRAPHICS TEXT. In Scaled Mode, text is initially written into the graphics memory. The size is fixed to allow for 35 lines of text. The text angle is set at 0 degrees and unslanted. The text origin is set to the left and bottom. These settings allow the "Page Full" feature to work properly and existing software to run without changes. If you do not require the Page Full feature, you can not change the text settings. You can redirect the text to the alphanumeric memory.

UNSCALED MODE GRAPHICS TEXT. In Unscaled Mode, the text size is unchanged and graphics text mode is not initially turned on. Text is stored in the alphanumeric memory unless the graphics text mode is specifically enabled.

Patterned Vectors. Whereas the 4010 draws only solid vectors, the 4014 has five line types — solid, dotted, short-dash, long-dash, and dot-dash. Escape sequences for these line types are accepted in SCL 4014 mode. These sequences retain their HP functional definitions in SCALED (4010) mode. The following commands select the line type for vector drawing:

- E c '** for normal vectors or alphanumeric data
- E c a** for dotted line vectors
- E c b** for dot-dashed vectors
- E c c** for short-dashed vectors
- E c d** for long-dashed vectors

The sequences **E c <h through l>** and **E c <p through t>** can also be used to select line types. The terminal accepts the line type specified by the sequence but ignores the 4014 line-width definition (defocused and write-thru vectors are not supported on the HP terminal).

Variable Character Sizes. The terminal ignores 4014 commands for changing character size. Thus the sequences **E c 8**, **E c 9**, **E c :** and **E c ;** are not executed in 4014 Mode.

UNSCALED 4014 (UNS 4014). The same vector-drawing capabilities available in Scaled 4014 mode apply in this mode, but no scaling is performed. The 512 by 390 screen displays a window on the 4096 by 3120 surface (the window is 1/64 the total image). As in unscaled mode, the relocatable origin is used to specify the window's lower left corner. Alphanumeric characters are directed to alpha memory, as they are in unscaled mode.

INCREMENTAL POINT PLOT. Sending the terminal an RS control character sets incremental point plot operation. Commands for pen up (SP)(i.e., an ASCII "SPACE") or pen down (P) control vector drawing in this mode. The terminal draws a single dot as the graphics beam moves in one-point increments to the following directional commands: D - North; E - Northeast; A - East; I - Southeast; H - South; J - Southwest; B - West; F - Northwest.

POINT PLOT. An FS control code selects point plot mode operation. As HP 4010 line type 11, only the last point of a vector is drawn.

SPECIAL POINT PLOT MODE. The terminal displays vectors drawn in this mode, but does not vary the intensity of the graphics beam.

Programming Considerations

When **SCALED** or **SCL 4014** mode is selected via the Terminal Configuration menu, applying power to the terminal or executing a hard reset turns on the graphics cursor and positions it in the upper left-hand corner of the screen display, emulating the 4014 function. When 4014 mode is selected programmatically, the state of the cursor and pen is not changed.

When the Global Configuration Menu **POWER ON** parameter is set to **COMPUTER**, P.A.M. resets **SCALE/SCALE 4014** to **OFF** in the Terminal Configuration Menu.

8-BIT MODE. Characters sent to the terminal in 4014 mode have their parity bit cleared automatically. Thus the **ROMAN** extension character set is not accessible in 8-Bit and Tektronix-modes. 7-bit mute processing retains its normal function.

Table D-14. Coding of Compatibility Mode Graphics Data

X or Y Coordinate																Low Order Y DEC.	Low Order X DEC.	ASCII
0	32	64	96	128	160	192	224	256	288	320	352	384	416	448	480	96	64	@
1	33	65	97	129	161	193	225	257	289	321	353	385	417	449	481	97	65	A
2	34	66	98	130	162	194	226	258	290	322	354	386	418	450	482	98	66	B
3	35	67	99	131	163	195	227	259	291	323	355	387	419	451	483	99	67	C
4	36	68	100	132	164	196	228	260	292	324	356	388	420	452	484	100	68	D
5	37	69	101	133	165	197	229	261	293	325	357	389	421	453	485	101	69	E
6	38	70	102	134	166	198	230	262	294	326	358	390	422	454	486	102	70	F
7	39	71	103	135	167	199	231	263	295	327	359	391	423	455	487	103	71	G
8	40	72	104	136	168	200	232	264	296	328	360	392	424	456	488	104	72	H
9	41	73	105	137	169	201	233	265	297	329	361	393	425	457	489	105	73	I
10	42	74	106	138	170	202	234	266	298	330	362	394	426	458	490	106	74	J
11	43	75	107	139	171	203	235	267	299	331	363	395	427	459	491	107	75	K
12	44	76	108	140	172	204	236	268	300	332	364	396	428	460	492	108	76	L
13	45	77	109	141	173	205	237	269	301	333	365	397	429	461	493	109	77	M
14	46	78	110	142	174	206	238	270	302	334	366	398	430	462	494	110	78	N
15	47	79	111	143	175	207	239	271	303	335	367	399	431	463	495	111	79	O
16	48	80	112	144	176	208	240	272	304	336	368	400	432	464	496	112	80	P
17	49	81	113	145	177	209	241	273	305	337	369	401	433	465	497	113	81	Q
18	50	82	114	146	178	210	242	274	306	338	370	402	434	466	498	114	82	R
19	51	83	115	147	179	211	243	275	307	339	371	403	435	467	499	115	83	S
20	52	84	116	148	180	212	244	276	308	340	372	404	436	468	500	116	84	T
21	53	85	117	149	181	213	245	277	309	341	373	405	437	469	501	117	85	U
22	54	86	118	150	182	214	246	278	310	342	374	406	438	470	502	118	86	V
23	55	87	119	151	183	215	247	279	311	343	375	407	439	471	503	119	87	W
24	56	88	120	152	184	216	248	280	312	344	376	408	440	472	504	120	88	X
25	57	89	121	153	185	217	249	281	313	345	377	409	441	473	505	121	89	Y
26	58	90	122	154	186	218	250	282	314	346	378	410	442	474	506	122	90	Z
27	59	91	123	155	187	219	251	283	315	347	379	411	443	475	507	123	91	[
28	60	92	124	156	188	220	252	284	316	348	380	412	444	476	508	124	92	\
29	61	93	125	157	189	221	253	285	317	349	381	413	445	477	509	125	93]
30	62	94	126	158	190	222	254	286	318	350	382	414	446	478	510	126	94	^
31	63	95	127	159	191	223	255	287	319	351	383	415	447	479	511	127	95	_
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	DEC.		
SP	!	..	#	\$	%	&	'	()	*	+	,	-	.	/	ASCII		
High Order X & Y																		

Table D-14. Coding of Compatibility Mode Graphics Data (Continued)

X or Y Coordinate																				Low Order Y		Low Order X																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
																				DEC.	ASCII	DEC.	ASCII																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
512	544	576	608	640	672	704	736	768	800	832	864	896	928	960	992	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
513	545	577	609	641	673	705	737	769	801	833	865	897	929	961	993	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439	1440	1441	1442	1443	1444	1445	1

Graphics Status

You can request graphics status information in addition to normal terminal status data. All graphics status requests are initiated by sending an `Ec * s` followed by a single parameter (1 through 12) and terminated by a caret (^). The single parameter selects the desired status block. If an invalid parameter is used, the terminal responds with its ID (see Device ID Request, parameter=1).

Graphics Status request: `Ec * s <parameter> ^`

where `Ec * s` is the graphics status escape sequence.

`<parameter>` is 1-12 and selects one of twelve blocks of graphics status data.

The graphics status blocks that can be requested are listed in Table D-15 together with the format of their terminal's response. Detailed descriptions of each status request are found in the following paragraphs.

The terminal responds with one or more bytes of status information followed by a block terminator. All status information is in ASCII format, with commas as separators. Coordinates are returned in a fixed format consisting of a sign and five digits. Leading zeros are used as required to provide a fixed number of digits (ie +00100, -01234). This allows you to use simple input statements without needing to mask or shift bits.

If DC1 handshake protocol is enabled (ie, "N0" is entered in the "InHndShk(G)" and "InhDC2(H)" fields of the Terminal Configuration menu), the status block is not actually sent until receipt of a DC1 character. If the DC1 character is used, only one graphics status request can be enabled while the terminal is waiting for a DC1. When the DC1 is received, the last graphics status block requested is sent to the terminal.

While the terminal is waiting for the DC1, the Device Status Pending bit is set.

The terminal's configuration defines the terminating character sent following the status block (CR, CR-LF, or RS). Graphics status requests turn on an echo suppress mode in the terminal (only if the graphics option is installed). This prevents information echoed back from the computer from being displayed on the screen. Once a graphics status block has been sent, characters received by the terminal are not displayed until one of the following control characters is received: BELL, BS, CR, EC, GS, HT, LF, RS, US, VT. With the exception of CR and LF, the terminating control code itself is executed.

The terminal expects the status information to be echoed and uses the terminating control character to turn off the suppress echo mode. If the computer does not echo the status, a suitable control character must be returned to the terminal to turn off echo suppression.

The graphics status blocks are shown in Table D-15.

Table D-15. Graphics Status Requests

Parameter	Request	Response
1	Read device I.D.	2623A OR 150A
2	Read current pen position	<X>, <Y>, <PEN>
3	Read graphics cursor position	<X>, <Y>
4	Read graphics cursor position with wait	<X>, <Y>, <KEY>
5	Read display size	<LLX>, <LLY>, <URX>, <URY>, <MMX>, <MMY>
6	Read device capabilities	<b1>, <b2>, . . . <b15>, <b16>
7	Read graphics text status	<X size>, <Y size>, <origin>, <angle>, <slant>
8	Read zoom status	001 . . 0
9	Read relocatable origin	<X>, <Y>
10	Read Reset status	<RESET>, <b1>, . . . <b6>, <b7>
11	Read area shading capability	1, 8, 8
12	Read dynamics capability	1, 1

Read Device ID (Parameter=1)

When you request a device ID the terminal responds with its generic Hewlett Packard model number, 2623A or 150A.

Device ID Request: `E c * s 1 ^`

The terminal responds: `2623A <terminator>`
or
`150A <terminator>`

Read Current Pen Position (Parameter=2)

The pen position and status are returned as a string of ASCII characters.

Pen Position Request: `E c * s 2 ^`

Where <X> = X coordinate
<Y> = Y coordinate
<Pen> = Pen state, 0 = pen up, 1 = pen down

For example, assume that the pen is at 360,80, the pen is up, and the terminal is set for the DC1 handshake, with CR as the terminator:

The computer sends: `Ec * 5 2 ^DC 1`

X coordinate Pen state

The terminal responds: `+00360, +00080, 0 CR`

Y coordinate

Read Graphics Cursor Position (Parameter=3)

The graphics cursor position is returned as a string of ASCII characters.

Read Graphics Cursor Request: `Ec * 5 3 ^`

The terminal responds: `<X> = X coordinate`

`<Y> = Y coordinate`

When the cursor is positioned in the lower left corner of the screen, the terminal's response is:

`+00000, +00000 CR`

Read Cursor Position With Wait (Parameter=4)

This request allows the user to position the cursor, then strike the key to return the position. The ASCII decimal code for the key stroke is also returned (not the actual character). The code is returned as three digits. For example, striking an uppercase A returns 065, the ASCII decimal code for an uppercase A. Only ASCII character keys generate a response (ie, ROLL UP, ROLL DOWN, etc., are ignored). The graphics cursor is turned on if it is not already on. If an escape sequence is received by the terminal after it has received the READ CURSOR with WAIT command and before a key is struck, the READ CURSOR command is aborted. The new sequence is executed instead.

Read Graphics Cursor with Wait Request: `Ec * 5 4 ^`

The terminal responds: `<X>, <Y>, <key code> <terminator>`

where `<X>` = X coordinate

`<Y>` = Y coordinate

`<key code>` = Decimal value of key struck

For example, if you position the cursor at the lower left corner of the screen then press the "A" key, the terminal responds:

`+00000, +00000, 065 CR`

The position bytes are ordered as in the read pen request (Parameter 2).

Read Display Size (Parameter=5)

This request returns the number of displayable units in the X and Y axes. It also returns the number of units per millimeter in the display. This request allows you to scale data for use on graphics devices with varying display area sizes.

Read Display Size Request: `Ec * s 5 ^`

The terminal

responds: `<LLX>,<LLY>,<URX>,<URY>,<MMX>,<MMY><terminator>`

Where: `<LLX>,<URX>` = Lower left and upper right X coordinates
`<LLY>,<URY>` = Lower left and upper right Y coordinates
`<MMX>,<MMY>` = Number of units per millimeter in the X and Y axes (five digits and a decimal point)

The terminal always returns a fixed response. The lower left corner has coordinates of 0,0. The upper right corner has coordinates of 511,389. There are approximately 2 units per millimeter in each axis.

Terminal response: `+00000,+00000,+00511,+00389,00003.,00003.<terminator>`

Read Device Capabilities (Parameter=6)

The device capabilities request returns a list of graphics plotting features available in the terminal. This allows you to use one program for a variety of graphics devices. Not all the features listed are available in the terminal. The absence of a feature is indicated by a zero (0). If a feature is present, it may be necessary to send an additional request to determine the exact capabilities present. Where multiple response values are possible, the terminal's standard response is enclosed in triple stars.

Drive Capability Request: `Ec * s 6 ^`

The terminal responds:

⟨b1⟩,⟨b2⟩,⟨b3⟩,⟨b4⟩,...⟨b16⟩⟨terminator⟩

where: ⟨b1⟩ = Clear display

 0 = no clear

 1 = paper advance

 2 = clear (total erase)

 3 = partial clear by area

⟨b2⟩ = Number of Pens (1)

⟨b3⟩ = Color Capability

 0 = black or white

 1 = gray levels

⟨b4⟩ = Color Level Capability (0)

 "0" means no color

⟨b5⟩ = Area Shading

 0 = no

 1 = yes (see Read Area Shading Capability)

⟨b6⟩,⟨b7⟩ = Not used (0,0)

⟨b8⟩ = Dynamic Modification

 0 = no

 1 = (see Read Modification Capability)

⟨b9⟩ = Graphic Character Size

 0 = fixed

 1 = Integer multiples of the basic cell size

⟨b10⟩ = Graphics Character Angles

 0 = fixed

 1 = Multiples of 90 degrees

 2 = multiples of 45 degrees

 3 = any angle

⟨b11⟩ = Graphics Character Slant

 0 = fixed

 1 = 45 degrees

 2 = any angle

⟨b12⟩ = Dot-Dash Line Patterns

 0 = none

 1 = predefined only

 2 = user-defined and predefined

⟨b13⟩-⟨b16⟩ = Not Used (0,0,0,0)

The terminal always responds:

3,1,0,0,1,0,0,1,1,1,1,2,0,0,0,0,⟨terminator⟩

Read Graphics Text Status (Parameter=7)

The terminal returns the current text size, orientation, slant, and type of justification. Refer to Graphics Text in this section for a description of graphics text characteristics.

Read Graphics Text Request: `Ec * s 7 ^`

The terminal returns: `<X size>,<Y size>,
<origin>,<angle>,<slant><terminator>`

where: `<X size>` = X dimension of the character cell (sign plus 5 digits)

`<Y size>` = Y dimension of the character cell (sign plus 5 digits)

`<origin>` = Relative position of text to cursor (see text origin command)(1 digit)

`<angle>` = Text angle 0, 90, 180, or 270 (5 digits and a decimal point)

`<slant>` = 00000. or 00027. degrees

Sample terminal response:

`+00007,+00010,1,00090.,00027.CR`

Read Zoom Parameter (Parameter=8)

This request returns the zoom setting. Since the terminal does not have the zoom feature, it always returns constant values.

Read Zoom Status Request: `Ec * s 8 ^`

The terminal responds:

`<zoom size>,<zoom on/off><terminator>`

where: `<zoom size>` = 001.

`<zoom on/off>` = 0 for Off

This response is always: `001.,0CR`

Read Relocatable Origin (Parameter=9)

The position of the relocatable origin is returned as X and Y coordinates.

Read Relocatable Origin Request: $E_c * s 9 ^\wedge$

The terminal responds: $\langle X \text{ coordinate} \rangle, \langle Y \text{ coordinate} \rangle$
 $\langle \text{terminator} \rangle$

With the origin set to the lower left corner of the screen, the terminal responds:

+00000,+00000CR

Read Reset Status (Parameter=10)

You can determine whether or not the terminal has executed a full reset (or Power On) since the last time reset status was checked. This tells whether or not you need to reestablish terminal settings or images before resuming terminal functions. An additional seven bytes are returned but are not used.

Read Reset Status Request: $E_c * s 10 ^\wedge$

The terminal responds:

$\langle \text{reset} \rangle, \langle b1 \rangle, \langle b2 \rangle, \langle b3 \rangle, \langle b4 \rangle, \langle b5 \rangle, \langle b6 \rangle, \langle b7 \rangle \langle \text{terminator} \rangle$

where: $\langle \text{reset} \rangle = 0$ No full reset since last check
or

1 Terminal has been reset

$\langle b1 \rangle - \langle b7 \rangle = 0$ (not used)

Read Area Shading Capability (Parameter=11)

The area shading capability of the terminal can be read. These are fixed for the terminal.

Read Area Shading Request: $E_c * s 11 ^\wedge$

The terminal always responds: 2,8,8 $\langle \text{terminator} \rangle$

The "2" indicates that the area shading can be a polygon. The first "8" indicates that the shading pattern is 8 units wide. The second "8" indicates that the shading pattern is 8 units high.

Read Graphics Modification Capabilities (Parameter=12)

You can read the terminal's dynamic graphics capabilities. This is the ability of the terminal to change selected portions of the display. These are fixed for the terminal.

Read Graphics Modification Request: `Ec * s 12 ^`

The terminal always responds: `1,1 <terminator>`

These two bytes indicate that the terminal has selective erase and complement capabilities.

Any Other Parameter

Any other parameter that has not been assigned causes the terminal ID to be returned. This is to prevent an invalid status request from tying up the requesting computer while waiting for a response.

The terminal responds: `2623A <terminator>`

or

`150A <terminator>`

The following BASIC program can be used to read all HP 150 status values:

```
10 REM .....
20 REM *   THIS PROGRAM ILLUSTRATES READING ALL HP150 STATUS VALUES   *
30 REM .....
40 REM .....STATUS VALUES FROM 1 TO 12 WILL BE READ
50 PRINT " A KEYBOARD RESPONSE IS REQUIRED AFTER THE THIRD RESPONSE "
60 FOR I=1 TO 12
70 PRINT CHR$(27);"s";I;"^";CHR(17)           'COMMAND TO READ STATUS
80 LINE INPUT A$                               'READ RESPONSE
90 PRINT "STATUS OF PARAMETER ";I;" IS "A$     'DISPLAY RESPONSE
100 NEXT I
110 STOP
```

GRAPHICS CONTROL ESCAPE SEQUENCES

Graphics Display Control

The following escape sequence controls the graphics display.

Ec *d <z> Performs the indicated action <z> on the graphics display.

<u>z</u>	<u>ACTION</u>
a	Clear graphics memory
b	Set graphics memory
c	Turn on graphics display
d	Turn off graphics display
e	Turn on alphanumeric display
f	Turn off alphanumeric display
k	Turn on graphics cursor
l	Turn off graphics cursor
m	Turn on rubber band line
n	Turn off rubber band line
<x>, <y> o	Move graphics cursor to horizontal position <x> and vertical position <y> (relative to the origin)
<x>, <y> p	Move graphics cursor to horizontal position <x> and vertical position <y> (relative to its present location)
q	Turn on alphanumeric cursor
r	Turn off alphanumeric cursor
s	Turn on graphic text mode
t	Turn off graphics text mode
z	No operation

Graphics Label Transmission

This escape sequence is used for transmission of a graphics text label from a program to the terminal.

Ec *l <text> The characters contained in <text> are printed
CR, CR LF, on the display starting at the pen position.
LF CR, or LF

Vector Drawing

The following escape sequences are used to draw vectors.

Ec *m <x>a Selects drawing mode <x>.

<u>x</u>	<u>MODE</u>
0	No effect
1	Clear
2	Set
3	Complement
4	Jam

Ec *m <x>b Selects line type <x>.

<u>x</u>	<u>LINE TYPE</u>	<u>x</u>	<u>LINE TYPE</u>
1	Solid line	7	Line #4
2	User line pattern	8	Line #5
3	Current area pattern	9	Line #6
4	line #1	10	Line #7
5	line #2	11	Point plot
6	line #3		

Ec *m <x>
<y> c

Defines an eight-bit segment of line pattern and a scale; where:

<x> is a number from 0 to 255 which, when converted to its binary form, illustrates the segment of line pattern.

<y> is a number from 0 to 255 which indicates the number of times the line pattern should be repeated.

Ec *m <a>
<c> <d> <e>
<f> <g> h <d>

Defines an 8×8 pattern where <a> through <h> are numbers from 0 through 255 which, when converted to their binary values and stacked, illustrate the pattern.

Ec *m <x1>,
<y1>, <x2>,
<y2> >e

Defines a rectangular area to be filled, where <x1>, <y1> and <x2>, <y2> define the rectangle located with respect to the absolute origin.

Ec *m <x1>,
<y1>, <x2>,
<y2> f

Defines a rectangular area to be filled, where <x1>, <y1> and <x2>, <y2> define the rectangle with respect to the relocatable origin.

Ec *m <x>,
<y> j

Locates the relocatable origin at coordinates <x>, <y> with respect to the absolute origin.

Ec *m <x> g

Selects area pattern "x":

<u>x</u>	<u>AREA PATTERN</u>
1	Solid area fill.
2	User-defined area fill (default).
3	Predefined pattern 0 (short dashed hatching).
4	Predefined pattern 1 (long dashed hatching).
5	Predefined pattern 2 (hatching).
6	Predefined pattern 3 (cross hatching).
7	Predefined pattern 4 (fine cross hatching).
8	Predefined pattern 5 (medium checkerboard.)
9	Predefined pattern 6 (fine checkerboard, 1:1 blend).
10	Predefined pattern 7 (3:1 blend).

Ec *m <x>h

Set area boundary pen <x>; where <x> is an integer in the range -32767 through 32767. The three low bits of the binary form of the integer is used to select the pen (0...7).

Ec *m k

Locates the relocatable origin at the current pen position.

Ec *m l

Locates the relocatable origin at the graphics cursor position.

Ec *m <x>m

Sets the graphics text size to <x>, where <x> is a number from 1 to 8.

Ec *m <x>n

Sets the graphics text orientation to <x>.

<u>x</u>	<u>ROTATION (DEGREES)</u>
1	0
2	90
3	180
4	270

Ec *m o

Turns on text slant.

Ec *m p

Turns off text slant.

Ec *m <x>q

Sets the origin of graphics text at location <x> on the display.

<u>x</u>	<u>LOCATION</u>	<u>x</u>	<u>LOCATION</u>
0	left/baseline	5	center/middle
1	left/bottom	6	center/top
2	left/middle	7	right/bottom
3	left/top	8	right/middle
4	center/bottom	9	right/top

Ec *m r

Set graphics defaults:

PARAMETER	DEFAULT
* Pen Condition	Down
* Line Type	1 (solid)
* Drawing Mode	2
* User Defined Line Pattern	255,1
* Area Fill Type	2 (User Defined Pattern)
* User Defined Area Fill Pattern	255,255..... (Solid)

PARAMETER	DEFAULT
* Boundary Pen	Off
* Graphics Text	Off
* Text Size	1
* Text Direction	1
* Text Origin	1 (left, bottom)
* Text Slant	Off
Relocatable Origin	0,0
Alpha Video	On
Graphics Video	On
Alpha Cursor	On
Graphics Cursor	Off
Graphics Cursor Address	0,0
Rubberband Line	Off
Compatibility Mode:	
Page Full Straps	0 (Out)
GIN Strap	0 (CR Only)

NOTE

Parameters marked with an asterisk are those affected by the sequence
Ec *m 1 r.

Ec *m 1 r	Sets the graphics defaults which are marked with an asterisk in the list above.
------------------	---------------------------------------------------------------------------------

Ec *m z	No operation.
----------------	---------------

Plotting Commands

This escape sequence is used in plotting vectors.

Ec *p <x> Performs action <x>

<u>x</u>	<u>ACTION</u>
a	Lift the pen
b	Lower the pen
c	Use graphics cursor position as new point
d	Draw a point at the current pen position and lift the pen
e	Set relocatable origin at the current pen position
f	Data is ASCII absolute
g	Data is ASCII incremental
h	Data is ASCII relocatable
i	Data is absolute
j	Data is short incremental
k	Data is incremental
l	Data is relocatable
s	Start area fill
t	End area fill
z	No operation

Graphics Status

This escape sequence reads the graphics status.

`Ec *s <x>^`

Reads status type `<x>`.

<u>x</u>	<u>STATUS</u>
1	Terminal I.D.
2	Pen position
3	Graphics cursor position
4	Read cursor position and wait for key
5	Display size
6	Graphics capabilities
7	Graphics text status
8	Read zoom status
9	Relocatable origin
10	Reset status
11	Area shading
12	Dynamics

Compatibility Mode

These escape sequences are used in Compatibility mode.

Ec *t <x>a	Selects graphics terminator.
<u>x</u>	<u>TERMINATOR</u>
0	CR
1	CR EOT
2	None
Ec *t <x>b	Sets or clears Page Full Break strap.
<u>x</u>	<u>ACTION</u>
0	Clear
1	Set
Ec *t <x>c	Sets or clears Page Full Busy strap.
<u>x</u>	<u>ACTION</u>
0	Clear
1	Set
Ec *t <x>d	Sets or clears 4014 mode:
<u>x</u>	<u>ACTION</u>
0	4010 mode
1	4014 mode
Ec *t z	No operation.
Ec *w r	Graphics hard reset.

INDEX

7-bit mode.....	6-6
8-bit mode.....	6-7

A

absolute addressing.....	2-31
absolute coordinate position.....	5-64, 5-81
absolute display coordinates.....	5-15
absolute format.....	5-61
advance line.....	2-47
advance page.....	2-47
AGIOS.....	2-21, 3-1
AGIOS calls.....	3-1, 3-3
AGIOS function calls.....	1-3, A-1, B-1/B-2
AGIOS function code.....	4-1, B-3
AGIOS functions.....	3-1, 4-33, B-10, C-1
AGIOS touch functions.....	B-1
AGIOS video functions.....	C-1
AIOS.....	3-2, 4-1
alphanumeric cursor.....	5-17/5-18
alphanumeric display memory.....	5-3
alphanumeric display.....	5-9/5-10
alphanumeric memory.....	2-25
alphanumeric screen size.....	5-103
alternate character sets.....	2-26
application softkey label.....	4-22, 4-24, 4-46
application softkeys.....	2-4, 4-21, 4-49, 4-67
area fill pattern.....	5-29
area fills.....	D-16/D-22
area pattern.....	5-74
area shading capability.....	5-99
ASCII control characters.....	5-55
ASCII data transfer.....	2-50
ASCII fields.....	2-14, 2-18, 4-66
ASCII keys.....	4-46
ASCII space.....	5-55
auto line feed.....	2-43

B

BASIC compiler.....	B-12
batch function call.....	4-3
binary data transfer.....	2-49
block mode transfer.....	2-42
block mode.....	2-40, 4-48
block transfers.....	2-40, 2-60
block-mode transmissions.....	2-43
block-transfer-priorities.....	2-61
boundary pen.....	5-36, 5-74/5-76, 5-84/5-88

Index

boundary.....5-35/5-36
break key.....6-11

C

cell size.....5-50
character set.....2-23, 4-6
clear area.....4-12
clear mode.....5-22, D-12
COM1.....6-1/6-2, 6-17
COM2.....6-1/6-2, 6-17
command completion-status.....2-59
compatibility mode, Graphics.....D-47/D-55
complement mode.....5-22/5-23, D-12
compress characters mode.....2-52
computer mode.....4-34, 4-48, 6-10/6-11
control functions.....4-25
cooked mode.....4-61/4-62, 6-17
copy a selected line.....2-48
copy all.....2-48
copy page.....2-48
current cursor position.....5-16, 5-71, 5-91
current pen position.....5-13, 5-61, 5-63/5-65, 5-71/5-73, 5-81/5-83
cursor positioning.....2-33
cursor types.....4-25
cursor-position sensing.....2-59
cursor-relative addressing.....2-31
cursor.....2-31

D

data comm control function.....6-3
data comm function code.....6-4
data comm functions.....6-5, 6-15
data comm programs.....6-1
data comm transparency mode.....6-8
data communications.....6-1
data logging mode.....2-47
data operations.....2-45
data transfer.....2-45
data word.....4-63/4-67
define area.....4-7
defined area.....4-5, 4-17
destination device.....2-45
device id.....4-63
device status.....2-59
device-status request.....2-75
display.....2-22
display control functions.....5-4
display enhancements.....2-22, 2-26, 4-6, 4-29
display memory.....2-31
display screen.....2-31
display structure.....2-30
draw mode.....5-51
drawing mode.....5-74
dynamic graphics capabilities.....5-100

E

enhance area.....	4-13
escape sequence processor.....	1-2
escape sequences.....	1-2, 2-1, A-1
escape sequences - Graphic Control.....	D-66

F

fast AGIOS entry.....	4-33
fields.....	2-11
form feed.....	2-47
format mode.....	2-38

G

GIOS.....	3-2, 5-1/5-2
GIOS function code.....	5-1
graphics attributes.....	5-95
graphics character, creating.....	5-49
graphics character set.....	5-40, 5-49, 5-57
graphics character set, re-defining.....	5-53
graphics compatibility mode.....	D-47/D-55
graphics control escape sequences.....	D-66
graphics control functions.....	5-40, D-1, D-66
graphics cursor.....	5-11/5-12, 5-15/5-16, 5-39
graphics cursor control.....	D-7
graphics default parameters.....	D-24
graphics display.....	5-7/5-8, D-2
graphics display control.....	D-7
graphics display memory.....	5-3, 5-5, 5-6
graphics drawing modes.....	D-9/D-13
graphics functions in display functions mode.....	D-36
graphics hard copy operations.....	D-38
graphics memory.....	5-22
graphics pad mode.....	4-50
graphics parameters.....	5-102
graphics plotting.....	5-61
graphics plotting functions.....	5-62
graphics plotting sequences.....	D-25/D-29
graphics screen size.....	5-103
graphics settings.....	5-93
graphics status.....	5-89, D-58
graphics status functions.....	5-89
graphics text.....	5-40, D-24/D-46
graphics text functions.....	5-40
graphics text label.....	5-48
graphics text mode.....	5-19
graphics text orientation.....	5-44
graphics text origin.....	5-47
graphics text sequence.....	5-40
graphics text size.....	5-41
graphics vector drawing mode.....	5-21

Index

H

handshaking.....2-42, 2-60, 6-2
high-level languages.....B-1, C-1
HP Roman-8 extensions.....5-55
HPIB device.....2-50

I

ignore.....4-60
incremental coordinate position.....5-69, 5-79, 5-82
incremental format.....5-61
intercept.....4-60
internal printer.....2-52

J

jam mode.....5-23, D-12

K

key characteristics.....4-46, 4-55, 4-57
keyboard.....2-20, 4-47
keyboard device id.....4-40
keyboard device id.....4-63
keyboard graphics functions.....D-3
keyboard intercept mode.....4-21, 4-47
keyboard intercept.....4-59
keyboard processing functions.....4-46
keyboard processing.....4-40
keycode.....4-58
keycode field.....4-40
keycode fields.....2-14, 4-66
keycode mode.....2-4, 2-8, 4-21, 4-40, 4-47, 4-59/4-60, 4-62/4-63, 4-66, 4-68
keycode table.....4-52

L

lift pen.....5-63
line feed.....2-47
line pattern and scale.....5-27
line type.....5-25
lower pen.....5-67

M

memory addressing.....2-31
memory lock.....2-33
metric format mode.....2-52
mode function keys.....2-34
modem.....6-10
move mode.....5-51
MS-pascal compiler.....B-5
multiple-character escape sequences.....2-1
multiple-character escape sequence.....2-2

N

non-interceptable functions.....	4-54
normal fields.....	2-15, 4-66
normal.....	4-60
numeric pad key, shifted.....	4-51
numeric pad mode.....	4-50

O

offset table.....	5-57
-------------------	------

P

parameters, graphics default.....	D-24
picture definition parameters.....	5-101
plotting sequences, graphics.....	D-25/D-29
point plot.....	5-26, 5-72
pointer table.....	5-53, 5-56
polygon boundary pen.....	5-84
polygon draw.....	5-81/5-83
polygon move.....	5-77/5-80
polygonal area fill.....	5-74
polygonal area fill pattern.....	5-34
Port 1.....	6-2, 6-19
Port 2.....	6-2, 6-19
primary terminal status.....	2-57, 2-62

Q

qualifier word.....	4-40, 4-54, 4-63, 4-66/4-67
---------------------	-----------------------------

R

Raw mode.....	4-21, 4-47, 4-59, 4-61, 6-17
read area.....	4-14
read data block.....	6-15
record mode.....	2-47
rectangular area fill.....	5-32/5-34
relocatable coordinate position.....	5-65, 5-70, 5-80, 5-83
relocatable format.....	5-33, 5-37/5-39, 5-61, 5-65, 5-70, 5-73, 5-80, 5-83, 5-97
relocatable origin.....	5-37/5-39, 5-73, D-22/D-23
remote/serial field.....	6-2
report format mode.....	2-52
reset status.....	5-98
row and column sensing.....	2-11
row/column reporting.....	2-15
row/column reports.....	4-66
rubber band line.....	5-13/5-14
rubber band line mode.....	D-28

S

screen-relative addressing.....	2-31
secondary terminal status.....	2-59, 2-67
selective erase.....	5-23
send data block.....	6-13
serial ports.....	6-1
set mode.....	5-22/5-23
shift area.....	4-16
shift bits field.....	4-64
shift bits.....	4-63
single graphics character.....	5-59
softkey fields.....	4-41
softkey labels.....	4-49
softkey touch field.....	4-41
softkey touch sensitivity.....	2-17
source device.....	2-46
special keys.....	4-46, 4-48, 4-57
status, graphics.....	D-58
status request.....	2-58, 2-60/2-61
system keys.....	2-4
system softkeys.....	4-21

T

terminal capabilities.....	2-69
terminal features.....	2-58
terminal ID status.....	2-61
terminal identification.....	2-58
terminal mode.....	2-1, 4-34, 4-48, 6-10/6-11
terminal status.....	2-63
text slant.....	5-45/5-46
toggle fields.....	2-14, 4-66
touch field.....	2-12, 2-14, 4-35
touchscreen.....	2-11, 4-38, 4-47, 4-66
touchscreen functions.....	4-35
touchscreen reporting.....	2-11, 4-44
transmit function key mode.....	2-20/2-21
two-character escape sequences.....	2-1, 2-27

U

user area fill pattern.....	5-29
user-definable softkeys.....	2-4/2-5, 2-17
user-defined keys.....	2-4
user-defined softkeys.....	2-8, 4-21, 4-49, 4-67

V

vector draw.....	5-68/5-70
vector drawing functions.....	5-21
vector drawing mode.....	5-22
vector line type.....	5-25
vector list.....	5-49/5-50, 5-52, 5-55, 5-57
vector move.....	5-64/5-66
vectors.....	5-61
vectors, graphics.....	D-29/D-35

video functions.....4-5
video intrinsics.....3-2, 4-5, 4-7

W

write area.....4-9
write line.....4-19

